МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

С.М. Порошин, В.М. Карташов,
В.В. Усик, Р.І. Цехмістро, І.С.Бєліков

ТЕХНОЛОГІЇ СТВОРЕННЯ СКЛАДОВИХ МУЛЬТИМЕДІЙНОГО КОНТЕНТУ. АНІМАЦІЯ ТА WEB-АНІМАЦІЯ

Навчальний посібник
для студентів усіх форм навчання за спеціальністю
123 «Комп'ютерна інженерія»

Затверджено
редакційно-видавничою радою
університету,

протокол № 3 від 06.10.2021р.

Харків 2022

УДК 0004.928
П59

Рецензенти:
В. Д. Карпов, доктор техн. наук, професор,
Харківський національний університет Повітряних Сил імені Івана Кожедуба;
В. І. Чумаков, доктор техн. наук, професор,
Харківський національний університет радіоелектроніки

Рекомендовано вченою радою НТУ «ХПІ»
як навчальний посібник для студентів усіх форм навчання за спеціальністю
123 «Комп'ютерна інженерія»,
протокол № 11 від 26.11.2021 р.

П59 **Порошин С.М., Карташов В.М., Усик В.В., Цехмістро Р.І., Бєліков І.С.**
Технології створення складових мультимедійного контенту. Анімація та web-анімація / С. М. Порошин, В. М. Карташов, В. В. Усик, Р. І. Цехмістро, І. С. Бєліков. – Харків : НТУ «ХПІ», 2022. – 314 с.

ISBN 978-966-8689-59-1

У навчальному посібнику викладено основні принципи створення веб-анімації на підставі сучасних технологій JavaScript, CSS-3, html-5, які використовують сучасну бібліотеку canvas для малювання, а також фреймворки bootstrap, jQuery, anime.js, three.js, GSAP, SVG; розглянуто основні положення створення персонажів у 3DS MAX; основи полігонального моделювання та роботи зі сплайнами; операції та ієрархія об'єктів; вибір та налаштування освітлення та камер; технологій аімаціїї персонажів у 3DS MAX. Наведено приклади 2D- і 3D-анімаційних сценаріїв та короткі описи можливостей професійних бібліотек.

Призначено для студентів спеціальності 123 «Комп'ютерна інженерія» та інших технічних спеціальностей.

Іл. 184. Табл. 35. Бібліогр. 19 назв.

УДК 0004.928

ISBN 978-966-8689-59-1

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AJAX	–	(«Asynchronous Javascript And Xml») технологія звернення до сервера без перезавантаження сторінки;
API	–	програмний інтерфейс додатка, інтерфейс прикладного програмування (англ. Application programming interface);
Bootstrap	–	фреймворк для створення адаптивних веб-сайтів;
CANVAS	–	елемент HTML5, призначений для створення реєстрового двовимірного зображення за допомогою скриптів;
CSS	–	каскадні таблиці стилів таблиць Cascading Style Sheets;
DOM	–	(DocumentObject Model) об’єктна модель документа;
3DS MAX	–	тривимірний графічний редактор;
GLSL	–	(OpenGL Shading Language, Graphics Library Shader Language) мова високого рівня для програмування шейдерів;
GSAP	–	(Javascript Animation Built For Professionals) платформа для анімації;
HTML	–	гіпертекстова розмітка сторінки;
JQUERY	–	JavaScript бібліотека для обходу та маніпуляції елементами HTML документа і AJAX запитів;
JS	–	(Java Script) мова програмування;
JSON	–	тип даних для відправки та отримання структурованих даних, об’єктів;
PHP	–	(Hypertext Preprocessor) це поширена мова програмування загального призначення з відкритим вихідним кодом;
ReactJS-React	–	(іноді React.js або ReactJS) JavaScript-бібліотека з відкритим вихідним кодом для розробки призначених для користувача інтерфейсів;
SVG	–	(Scalable Vector Graphics) мова розмітки векторної графіки, що масштабується;
three.js	–	бібліотека тривимірної графіки та 3d-анімації, яка заснована на технології OpenGL Shading Language;
TweenMax.js	–	бібліотека, яка тепер є частиною GSAP-платформи анімації;
WebGL	–	(Web-based Graphics Library) багатоплатформовий API для 3D-графіки в браузері;
ДС	–	джерело світла;
МВО	–	мінімальна опукла оболонка;
ООП	–	об’єктно-орієнтоване програмування.

ВСТУП

Розвиток науки, суспільства, нових технологій зростає настільки швидкими темпами, що нові знання доволі швидко втрачають свою актуальність, застарівають. Поряд з цим швидкість зміни інформації у сучасному світі надзвичайно висока, тому гостро постає питання формування інформаційних ресурсів на основі інтеграції інформаційних технологій, які забезпечують активний вплив людини на ці дані в реальному масштабі часу.

Мультимедійні технології є тією навігаційною структурою, що забезпечує інтерактивність – можливість безпосередньої взаємодії з програмним ресурсом.

Мультимедіа – це спеціальна інтерактивна технологія, яка за допомогою технічних і програмних засобів забезпечує роботу з комп'ютерною графікою, текстом, мовленнєвим супроводом, високоякісним звуком, статичними зображеннями й відео.

Мультимедійний контент – електронні комбінації інформації, що містять текст, анімацію, відеодані, нерухомі зображення, аудіопотоки, які доступні в інтерактивному режимі. На web-сайтах, у додатках до друкованих видань сьогодні значна частина інформації подається у вигляді саме мультимедійних ресурсів.

Сьогодні використання Інтернет ресурсів набуло широкого застосування і є чи не головним джерелом у наданні користувачу різнопланової інформації із будь-якої сфери діяльності. Саме тому використання широкого спектра можливостей мультимедійних систем є достатньо важливим для розробників Web-сторінок. Для створення якісного web-сайта з мультимедійним контентом одним із головних критеріїв є застосування відповідних графічних можливостей для створення хорошого дизайну та найзрозумілішого відображення інформації, яка подається Інтернет сторінкою.

Мультимедійні технології роботи з графічною інформацією застосовується при:

– створенні графіки, яку часто разом з анімацією визначають як реалістичну графіку, що широко використовується в таких галузях, як автоматизоване проектування, кіно, телебачення та комп'ютерні ігри;

– анімації у різних виглядах;

– поєднанні телевізійних та комп'ютерних зображень.

Структура навчального посібника передбачає послідовне ознайомлення читача з основами мови JavaScript для веб-розробок та її основних бібліотек і фрейверків, що дозволяє розглянути створення найпростіших, але вражаючих анімаційних ефектів з використанням CSS, що дозволяє оживити веб-сайт будь-якого призначення. Викладення матеріалу послідовно підводить читача до ознайомлення з бібліотеками професійної 3D-графіки для веб-розробок. Посібник не претендує на охоплення всіх сучасних технологій створення анімацій і комп'ютерних ігор, він має на меті ознайомити читача з основою створення анімаційних фрагментів, придатних до використання на веб-ресурсах різноманітного (не обов'язково розважального) призначення. Приділено увагу математичним алгоритмам комп'ютерної графіки, зокрема тріангуляції Делоне.

У посібнику розглянуті: основні положення створення персонажів у 3DS MAX; основи полігонального моделювання та роботи зі сплайнами; операції та ієрархія об'єктів; вибір та налаштування освітлення та камер; технологій аімаціїї персонажів у 3DS MAX.

Кожний розділ має приклади виконання окремих завдань, запитання для самоперевірки та індивідуальні завдання для вирішення.

Викладений у посібнику матеріал дозволить створювати якісний анімаційний контент як складову мельтимедіного контенту, що в свою чергу дозволить підвищувати швидкість засвоєння інформацій.

1. ЗАГАЛЬНІ ВІДОМОСТІ ПРО HTML-5, CSS-3

1.1. Поняття html

HTML-теги – основа мови HTML (в перекладі це гіпертекстова розмітка сторінки). Теги використовуються для розмежування початку і кінця елементів у розмітці.

Кожен HTML-документ складається з дерева HTML-елементів і тексту. Кожен HTML-елемент позначається початковим (відкриваючим) і кінцевим (закриваючим) тегом. Теги, які відкривають та закривають, містять ім'я тега. Теги поділяються на:

– порожні елементи <area>, <base>,
, <col>, <embed>, <hr>, <img>, <input>, <link>, <menuitem>, <meta>, <param>, <source>, <track>, <wbr>;

– елементи з неформатований текстом – <script>, <style>;

– елементи, які відображають звичайний текст – <textarea>, <title>;

– елементи з іншого простору імен – MathML и SVG.

Кожен тег має свої атрибути, яким у свою чергу присвоюються значення, наприклад

<p style="font: 12pt Courier"> – у цьому прикладі це текст з кеглем 12 точок і гарнітурою Courier</P>

Таблиця 1.1 – Короткий опис основних тегів

Тег	**Опис**
1	2
<! --...-->	Використовується для додавання коментарів.
<! DOCTYPE>	Оголошує тип документа і надає основну інформацію для браузера – його мову і версію.
<a>	Створює гіпертекстові посилання.
<abbr>	Визначає текст як абревіатуру або акронім. Пояснювальний текст задається за допомогою атрибута title.
<address>	Задає контактні дані автора/власника документа або статті. Відображається в браузері курсивом.
<area>	Являє собою гіперпосилання з текстом, що відповідає певній області на карті-зображенні або активну область всередині карти-зображення. Завжди вкладений всередину ті-га <map>.
<article>	Розділ контенту, який утворює незалежну частину документа або сайту, наприклад, стаття в журналі, запис у блозі, коментар.

Продовження таблиці 1.1

1	2
<ide>	Являє контент сторінки, який має непряме відношення до основного контенту сторінки / сайту.
<audio>	Завантажує звуковий контент на веб-сторінку.
<b>	Завантажує звуковий контент на веб-сторінку.
<base>	Задає базову адресу (URL). Це допоможе уникнути проблем під час перенесення сторінки в інше місце, оскільки усі посилання працюватимуть, як і раніше.
<bdi>	Ізолює уривок тексту, написаний мовою, в якій читання тексту відбувається справа наліво, від решти тексту.
<bdo>	Відображає текст у напрямку, зазначеному в атрибуті dir, перевизначаючи поточний напрямок написання тексту.
<blockquote>	Виділяє текст як цитату, застосовується для опису великих цитат.
<body>	Являє тіло документа (вміст, що не належить до метаданих документа).
*** ***	Перенесення тексту на новий рядок.
<button>	Створює інтерактивну кнопку. Всередину тега можна помістити вміст - текст або зображення.
<canvas>	Полотно-контейнер для динамічного відображення зображень, таких як прості зображення, діаграми, графіки тощо. Для малювання використовується скриптова мова JavaScript.
<caption>	Додає підпис до таблиці. Вставляється відразу після тега <table>.
<cite>	Використовується для вказівки джерела цитування. Відображається курсивом.
<code>	Являє фрагмент програмного коду, відображається шрифтом сімейства monospace.
<col>	Вибирає для форматування один або декілька стовпців таблиці, які містять інформацію одного типу.
<colgroup>	Створює структурну групу стовпців, що виділяє безліч логічно однорідних осередків.
<data>	Елемент використовується для зв’язування значення атрибута value, що подано у форматі машинозчитування та може бути оброблено комп’ютером, з вмістом тега.
<datalist>	Елемент-контейнер для списку елемента <input>. Варіанти значень поміщаються в елементи <option>.
<dd>	Використовується для опису терміна з тега <dt>.
<del>	Позначає текст, який видалено, перекреслюючи його.

Продовження таблиці 1.1

1	2
<details>	Створює інтерактивний віджет, який користувач може відкрити або закрити. Являє собою контейнер для контенту, видимий заголовок віджета поміщається в тег <summary>.
<dfn>	Визначає слово як термін, виділяючи його курсивом. Текст, що йде слідом, має містити розшифровку цього терміна.
<dialog>	Інтерактивний елемент, з яким взаємодіє користувач для виконання завдання, наприклад, діалогове вікно, інспектор або вікно. Без атрибута open непомітний для користувача.
*** ***	Тег-контейнер для розділів HTML-документа. Використовується для угруповання блокових елементів з метою форматування стилями.
<dl>	Тег-контейнер, всередині якого знаходяться термін і його опис.
<dt>	Використовується для завдання терміна.
******	Виділяє важливі фрагменти тексту, відображаючи їх курсивом.
<embed>	Тег-контейнер для вбудовування зовнішнього інтерактивного контенту або плагіна.
<fieldset>	Групує пов'язані елементи в формі, малюючи рамку навколо них.
<figcaption>	Тема/підпис для елемента <figure>.
<figure>	Самодостатній тег-контейнер для такого контенту як ілюстрації, діаграми, фотографії, приклади коду, зазвичай з підписом.
<footer>	Визначає завершальну область (нижній колонтитул) документа або розділу.
<form>	Форма для збору і відправки на сервер інформації від користувачів. Не працює без атрибута action.
<h1-h6>	Створюють заголовки шести рівнів для пов’язаних з ними розділів.
<head>	Елемент-контейнер для метаданих HTML-документа, таких як <title>, <meta>, <script>, <link>, <style>.
<header>	Секція для вступної інформації сайту або групи навігаційних посилань. Містить один або кілька заголовків, логотип, інформацію про автора.
<hr>	Горизонтальна лінія для тематичного поділу параграфів.
<html>	Кореневий елемент HTML-документа. Повідомляє браузеру, що це HTML-документ. Є контейнером для всіх інших HTML-елементів.

Продовження таблиці 1.1

1	2
<i>	Виділяє уривок тексту курсивом, не надаючи йому додаткового акценту.
<iframe>	Створює вбудований фрейм, завантажуючи в поточний HTML-документ інший документ.
<img>	Вбудовує зображення в HTML-документ за допомогою атрибута src, значенням якого є адреса вбудованого зображення.
<input>	Створює багатофункціональні поля форми, в які користувач може вводити дані.
<ins>	Виділяє текст підкресленням. Застосовується для виділення змін, що вносяться до документа.
<kbd>	Виділяє текст, який має бути введений користувачем з клавіатури, шрифтом сімейства monospace.
<label>	Додає текстову мітку для елемента <input>.
<legend>	Тема елементів форми, згрупованих за допомогою елемента <fieldset>.
*** ***	Елемент маркірованих або нумерованих списків.
<link>	Визначає відносини між документом і зовнішнім ресурсом. Також використовується для підключення зовнішніх таблиць стилів.
<main>	Контейнер для основного унікального вмісту документа. На одній сторінці має бути не більше одного елемента <main>.
<map>	Створює активні області на карті-зображенні. Є контейнером для елементів <area>.
<mark>	Виділяє фрагменти тексту, позначаючи їх жовтим фоном.
<meter>	Індикатор вимірювання в заданому діапазоні.
<meta>	Використовується для зберігання додаткової інформації про сторінку. Цю інформацію використовують браузери для обробки сторінки, а пошукові системи – для її індексації. У блоці <head> може бути декілька тегів <meta>, оскільки залежно від використовуваних атрибутів вони несуть різну інформацію.
<nav>	Розділ документа, що містить навігаційні посилання на сайті.
<noscript>	Визначає секцію, яка не підтримує сценарій (скрипт).
<object>	Контейнер для вбудовування мультимедіа (наприклад, аудіо, відео, Java-аплети, ActiveX, PDF і Flash). Також можна вставити іншу веб-сторінку в поточний HTML-документ. Для передачі параметрів вбудованого плагіна використовується тег <param>.
*** ***	Упорядкований нумерований список. Нумерація може бути числова або алфавітна.

Продовження таблиці 1.1

1	2
<***optgroup***>	Контейнер з заголовком для групи елементів <option>.
<***option***>	Визначає варіант / опцію для вибору в списку <select>, <optgroup> або <datalist>.
<***output***>	Поле для виведення результату обчислення, розрахованого за допомогою скрипта.
<***p***>	Параграфи в тексті.
<***param***>	Визначає параметри для плагінів, вбудованих за допомогою елемента <object>.
<***picture***>	Елемент-контейнер, що містить один елемент <img> і нуль або декілька елементів <source>. Сам по собі нічого не відображає. Дає можливість браузеру вибирати найбільш відповідне зображення.
<***pre***>	Виводить текст без форматування, зі збереженням прогалин і переносів тексту. Може бути використаний для відображення комп'ютерного коду, повідомлення електронної пошти і т.д.
<***progress***>	Індикатор виконання завдання будь-якого роду.
<***q***>	Визначає коротку цитату.

1.2. Поняття про CSS і селектори

HTML задає основну структуру веб-сторінки, а також вказує, які елементи на ній присутні. Самооформлення веб-сторінки, положення та вид елементів покладено на стилі або CSS (Cascading Style Sheets, каскадні таблиці стилів). Коли говорять про верстку веб-сторінок, йдеться про синергію HTML і CSS.

Сам HTML не становить окремого інтересу, через свою простоту і обмеженість. Також і CSS не грає окремої ролі, оскільки прив'язується до певних елементів коду і задає їх оформлення. Тому працюючи разом в одній зв'язці, вони перетворюють скромну сторінку в той документ, який придумав і намалював дизайнер. Таке взаємне посилення властивостей, підсумовуючий ефект і є синергією.

Будь-яка веб-сторінка це, по суті, комбінація HTML-коду і CSS-коду. Без основних знань цих технологій не вийде грамотно зверстати жоден документ. Тому даний параграф присвячений основам CSS і його застосуванню на практиці. Якщо ви вважаєте, що це вже вам відомо, можете пропустити даний розділ і перейти до наступного.

CSS (Cascading Style Sheets) – мова таблиць стилів, який дозволяє прикріплювати стиль (наприклад, шрифти і колір) до структурованих

документів (наприклад, документів HTML і додатків XML). Зазвичай CSS-стилі використовуються для створення і зміни стилю елементів веб-сторінок і призначених для користувача інтерфейсів, написаних мовами HTML і XHTML, але також можуть бути застосовані до будь-якого виду XML-документа, в тому числі XML, SVG і XUL. Відокремлюючи стиль подання документів від вмісту документів, CSS спрощує створення веб-сторінок і обслуговування сайтів.

CSS підтримує таблиці стилів для конкретних носіїв, тому автори можуть адаптувати подання своїх документів до візуальних браузерів, слухових пристроїв, принтерів, брайлівських пристроїв, кишенькових пристроїв і т.д.

Каскадні таблиці стилів описують правила форматування елементів за допомогою властивостей і допустимих значень цих властивостей. Для кожного елемента можна використовувати обмежений набір властивостей, інші властивості не впливатимуть на нього. Приклад показано на рис.1.1.

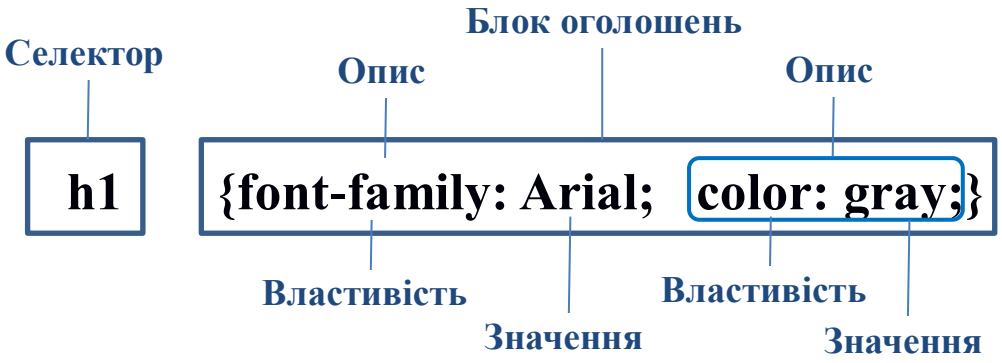


Рисунок 1.1 – Структура оголошення

Типи стилів

Розрізняють декілька типів стилів, які можуть спільно застосовуватися до одного документу. Це стиль браузера, стиль автора і стиль користувача.

Стиль браузера – це оформлення, яке за замовчуванням застосовується до елементів веб-сторінки браузером. Це оформлення можна побачити в разі «голого» HTML, коли до документу не додається жодних стилів. Наприклад, заголовок сторінки, що формується тегом <H1>, у більшості браузерів виводиться шрифтом із зарубками розміром 24 пункту.

Стиль автора – це стиль, який додає до документа його розробник.

Стиль користувача – це стиль, який може включити користувач сайту через налаштування браузера. Такий стиль має більш високий пріоритет і перевизначає вихідне оформлення документа.

Як селектор може виступати будь-який тег HTML, для якого визначаються правила форматування, такі як: колір, фон, розмір тощо. Правила задаються в такому вигляді.

Тег {свойство1: значення; властивість2: значення; ...}

Спочатку вказується ім'я тега, оформлення якого буде перевизначено, великими або малими символами, не має значення. Усередині фігурних дужок пишеться стильова властивість, а після двокрапки – його значення. Набір властивостей поділяється між собою крапкою з комою і може розташовуватися як в один рядок, так і в кілька.

Існують три типи селектора CSS:

- **звичайні**, де «p» в CSS націлюється на HTML-елементи <p>;
- **класи**, де «.intro» в CSS націлюється на елементи з атрибутом class = "intro";
- **ідентифікатори**, де «#logo» в CSS націлюється на елемент з атрибутом id = "logo".

До всіх цих селекторів можуть прикріплятися псевдокласи. Псевдоклас:

- визначає конкретний стан елемента;
- це ключове слово, яке починається з двокрапки.

Псевдоклас не може існувати сам по собі. Він має бути прикріплений до селектора. Псевдоклас визначатиме тільки певний стан цього селектора.

Синтаксис виглядає так: селектор:псевдоклас {}.

Прогалини між селектором і псевдокласом немає, щоб показати, що вони пов'язані один з одним.

Селектор: hover

Наприклад, типовим використовуваним псевдокласом є: hover, який застосовуватиме стиль, коли на цільовий елемент наводиться курсор мишки. Перевіримо це на посиланнях, розглянувши такий код:

```
<P> Наведіть курсор <a href="#"> на це посилання </a> і побачите, як воно стає червоним. </ P>
a { color: blue; }
a:hover { color: red; }
```

Селектор: visited

Цей псевдоклас націлюється на посилання, які вже були відвідані. За замовчуванням посилання відображаються синіми і при відвідуванні стають фіолетовими. Результати Google працюють так само.

```
<a href="https://www.google.com"> Google </a>
<a href="https://twitter.com"> Twitter </a>
a {color: dodgerblue; }
a: visited {color: rebeccapurple; }
```

Селектор: focus

Даний псевдоклас відбувається, коли елемент HTML отримує фокус. Це особливо корисно для полів форм.

```
.form-input {border: 2px solid grey; padding: 5px; }
.form-input: focus {background: lightyellow; border-color: blue; outline: none; }
```

Селектори: first-child і: last-child

Ці псевдокласи пов'язані з ієрархією в HTML. Вони націлюються на елементи HTML залежно від порядку, в якому з'являються в html коді. Наприклад,

```
<ul>
 <li> один </ li>
 <li> два </ li>
 <li> три </ li>
 <li> чотири </ li>
</ ul>;
```

в стильовому коді

```
li: first-child {background: greenyellow;}
li: last-child {background: lightsalmon;}.
```

У таблиці 1.2 наведено перелік псевдокласів, які найчастіше використовуються в ході створення кода.

Таблиця 1.2 – Список псевдо-класів

Псевдоклас	Призначення
1	2
:fullscreen	Застосовується до елементів, коли браузер знаходиться в повноекранному режимі.
:hover	Визначає стиль елемента з наведенням на нього курсора мишки, але при цьому елемент ще не активований.
:in-range	Застосовується до елементів форм, у яких введене користувачем значення знаходиться в заздалегідь заданому діапазоні. Сам діапазон встановлюється за допомогою атрибутів min і max, вони, відповідно, визначають мінімальне та максимальне значення.

Продовження таблиці 1.2

1	2
:indeterminate	Задає стиль для елементів форм, таким як прапорці та перемикачі, коли вони знаходяться в невизначеному стані.
:invalid	Застосовується до полів форми, вміст яких не відповідає вказаним типам.
:lang	Визначає мову, яка використовується в документі або його фрагменті.
:last-child	Задає стильове оформлення останнього дочірнього елемента в групі братських елементів.
:last-of-type	Задає правила стилів для останнього елемента певного типу в групі братських елементів.
:link	Застосовується до посилань, які ще не відвідувалися користувачем.
:not	Задає правила стилів для елементів, які не містять вказаний селектор.
:nth-child	Використовується для додавання стилю до елементів на основі нумерації в дереві елементів.
:nth-last-child	Використовується для додавання стилю до елементів на основі нумерації в дереві елементів, відлік ведеться з кінця.
:nth-last-of-type	Використовується для додавання стилю до елементів зазначеного типу на основі нумерації в дереві елементів, відлік ведеться від останнього елемента.
:nth-of-type	Використовується для додавання стилю до елементів зазначеного типу на основі нумерації в дереві елементів.
:only-child	Застосовується до дочірнього елементу, тільки якщо він є єдиним у свого батька.
:only-of-type	Застосовується до дочірнього елементу зазначеного типу, тільки якщо він єдиний у батька.
:optional	Застосовує стильові правила до поля форми, у якого не заданий атрибут required.
:out-of-range	Застосовується до полів форм, у яких введене користувачем значення виходить із заданого діапазону. Псевдоклас працює тільки для тих полів, де користувач може сам ввести значення, незважаючи на обмеження.
:placeholder-shown	Визначає стиль елементу <input> або <textarea>, який в даний момент відображає текст підказки, заданої атрибутом placeholder.
:read-only	Застосовується до полів форми, у яких заданий атрибут readonly.
:read-write	Застосовується до полів форми, доступних для зміни.
:required	Застосовує стильові правила до елементу input, у якого встановлений атрибут required.

Продовження таблиці 1.2

1	2
:target	Застосовується до цільового елементу, іншими словами, до ідентифікатора, який вказаний в адресному рядку браузера.
:valid	Застосовується до полів форми, вміст яких проходить перевірку в браузері на відповідність зазначеного типу.
:visited	Застосовується до посилань, вже відвіданих користувачем, і задає для них стильове оформлення.

Псевдоелементи

Псевдоелементи дозволяють задати стиль елементів, не визначених у дереві елементів документа, а також генерувати вміст, чого немає у вихідному коді тексту. Синтаксис використання псевдоелементів такий – Селектор: Псевдоелемент {Опис правил стилю}.

Спочатку йде ім’я селектора, потім пишеться двокрапка, після якого йде ім’я псевдоелемента. Кожен псевдоелемент може застосовуватися тільки до одного селектора, якщо потрібно встановити відразу декілька псевдоелементів для одного селектора, правила стилю мають додаватися до них окремо, як показано нижче.

.foo: first-letter {color: red}
.foo: first-line {font-style: italic}

В CSS3 щоб розрізняти псевдокласи і псевдоелементи, перед ім’ям псевдоелемента ставиться дві двокрапки (:: after).

Internet Explorer ігнорує подібний запис, інші браузери коректно її розуміють. В таблиці 1.3 навединий перелік псевдоелементів.

Таблиця 1.3 – Таблиця псевдоелементів

Псевдоелемент	**Призначення**
::after	Додає вміст після елемента
::before	Додає вміст до елемента
::first-letter	Стиль першої літери текстового блоку
::first-line	Визначає стиль першого рядка блокового тексту. Довжина цього рядка залежить від багатьох факторів, таких як шрифт, що використовуються, розмір вікна браузера, ширина блоку, мови і т.д.

Класи застосовують, коли необхідно визначити стиль для індивідуального елемента веб-сторінки або задати різні стилі для одного тега. У використанні спільно з тегами синтаксис для класів буде таким.

Тег.Ім'я класу {властивість1: значення; властивість2: значення; ...}

Усередині стилю спочатку пишеться бажаний тег, а потім, через крапку користувальницьке ім'я класу. Імена класів мають починатися з латинського символу і можуть містити символ дефіса (-) і підкреслення (_).

Використання кирилиці в іменах класів неприпустимо. Щоб вказати в коді HTML, що тег використовується з певним класом, до тегу додається атрибут «class» значенням якого виступає ім'я класу. Приклад нижче.

```
<!DOCTYPE html PUBLIC "- // W3C // DTD XHTML 1.0 Strict // EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns = "http://www.w3.org/1999/xhtml">
 <head>
 <meta http-equiv = "Content-Type" content = "text / html; charset = utf-8" />
 <title> Класи </ title>
 <style type = "text / css">
 p {/ * Звичайний абзац * /
 text-align: justify; / * Вирівнювання тексту за шириною * /
 }
 p.cite {/ * Абзац з класом cite * /
 color: navy; / * Колір тексту * /
 margin-left: 20px; / * Відступ зліва * /
 border-left: 1px solid navy; / * Кордон зліва від тексту * /
 padding-left: 15px; / * Відстань від лінії до тексту * /
 }
 </ style>
 </ head>
 <body>
 <p> Для штучного освітлення приміщення застосовуються люмінесцентні
лампи. Вони відрізняються високою світловою віддачею, тривалим
терміном служби, малою яскравістю світіння поверхні, близьким до
природного спектральним складом випромінюваного світла, що забезпечує
гарну передачу кольору.</ p>
 <p class = "cite"> Для виключення засвічення екрану дисплея світловими
потоками віконні прорізи забезпечені світлорозсіювальними шторами. </ p>
 </ body>
</ html>
```

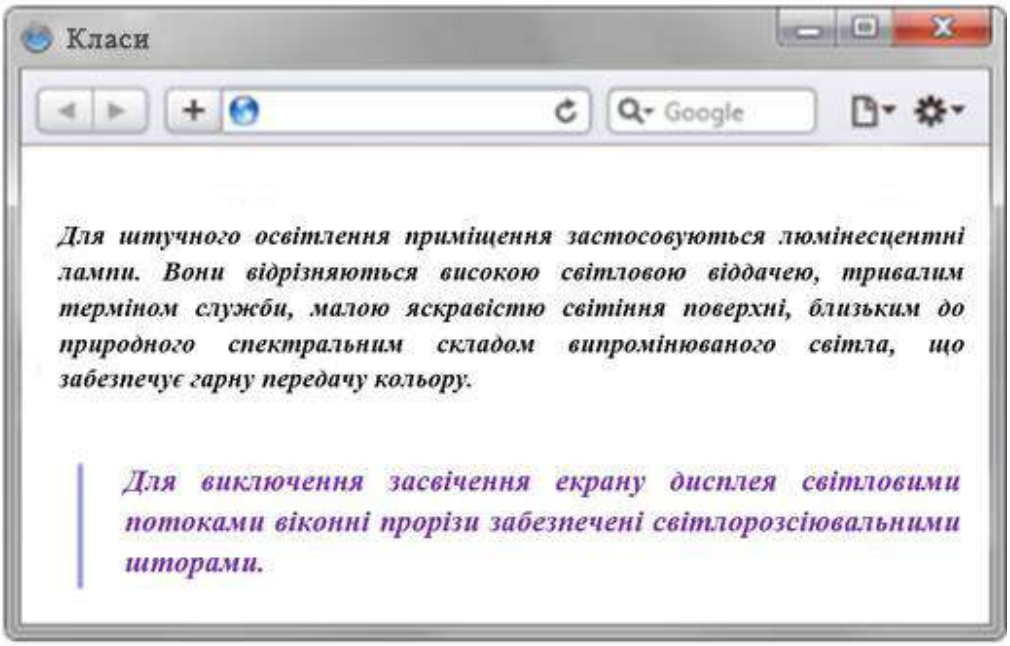


Рисунок 1.2 – Вид тексту, оформленого за допомогою стильових класів

Перший абзац вирівняний за шириною з текстом чорного кольору (цей колір задається браузером за замовчуванням), а наступний, до якого застосовано клас з ім’ям cite – відображається синім кольором і з лінією зліва.

Можна також використовувати класи і без зазначення тега. Синтаксис у цьому випадку буде такий.

.Ім’я класу {властивість1: значення; властивість2: значення; ...}

Групування

Під час створення стилю для сайту, коли водночас використовується безліч селекторів, можлива поява повторюваних стильових правил. Щоб не повторювати двічі одні й ті самі елементи, їх можна згрупувати для зручності подання та скорочення коду. У прикладі нижче показано звичайний запис, тут для кожного селектора наводиться свій набір стильових властивостей.

```
H1, H2, H3 {
 font-family: Arial, Helvetica, sans-serif;
}
H1 {
 font-size: 160%;
 color: # 003;
}
H2 {
 font-size: 135%;
```

```
 color: # 333;
}
H3 {
 font-size: 120%;
 color: # 900;
}
```

Спадкування

Спадкуванням називається перенесення правил форматування для елементів, що знаходяться всередині інших. Такі елементи є дочірніми, і вони успадковують деякі стильові властивості своїх батьків, усередині яких розташовуються. Розберемо спадкування на прикладі таблиці. Особливістю таблиць можна вважати чітку ієрархічну структуру тегів. Спочатку йде контейнер <table>, всередині якого додаються теги <tr>, а потім йде тег <td>. Якщо в стилях для селектора TABLE задати колір тексту, то він автоматично встановлюється для вмісту, приклад нижче.

```
<! DOCTYPE html PUBLIC "- // W3C // DTD XHTML 1.0 Strict // EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns = "http://www.w3.org/1999/xhtml">
 <head>
 <meta http-equiv = "Content-Type" content = "text / html; charset = utf-8" />
 <title> Спадкування </ title>
 <style type = "text / css">
 TABLE {
 color: red; / * Колір тексту * /
 background: # 333; / * Колір фону таблиці * /
 border: 2px solid red; / * Червона рамка навколо таблиці * /
 }
 </ style>
 </ head>
 <body>
 <table cellpadding = "4" cellspacing = "0">
 <tr>
 <td> Осередок 1 </ td> <td> Осередок 2 </ td>
 </ tr>
 <tr>
 <td> Осередок 3 </ td> <td> Осередок 4 </ td>
 </ tr>
 </ table>
 </ body>
</ html>
```

1.3. Переходи CSS

Щоб перехід мав місце, елемент має отримати зміну стану і для кожного стану мають бути визначені різні стилі. Найпростіший спосіб задати стилі для різних станів – це скористатися псевдокласом: hover,: focus,: active і: target. Є чотири властивості, пов'язаних з переходами в цілому, це transition-property, transition-duration, transition-timing-function і transition-delay. Не всі вони є обов'язковими для створення переходу, найбільш популярні перші три.

Вендорні префікси

Наведений вище код, як і інші зразки коду, не використовують вендорні префікси. Це зроблено навмисно в інтересах збереження фрагмента коду маленьким і зрозумілим. Щоб отримати найкращу функціональність у всіх браузерах, використовуйте префікси. Для довідки, версія з префіксами для коду вище матиме такий вигляд.

```
.box {
 background: # 2db34a;
 -webkit-transition-property: background;
 -moz-transition-property: background;
 -o-transition-property: background;
 transition-property: background;
 -webkit-transition-duration: 1s;
 -moz-transition-duration: 1s;
 -o-transition-duration: 1s;
 transition-duration: 1s;
 -webkit-transition-timing-function: linear;
 -moz-transition-timing-function: linear;
 -o-transition-timing-function: linear;
 transition-timing-function: linear;
}
.box: hover {
 background: # ff7b29;
}
```

Transition-property

Властивість transition-property визначає, які саме властивості змінюватимуться в поєднанні з іншими властивостями переходу. За замовчуванням змінюватися будуть всі властивості в різних станах. Проте, тільки властивості задані в значенні transition-property будуть порушені в будь-яких переходах. У наведеному вище прикладі властивість background

визначено в значенні transition-property. Тут background – єдина властивість, яка змінюватиметься протягом 1 секунди в лінійному вигляді. Будь-які інші властивості, включені коли змінюється стан елемента, але не додані в значення transition-property, не отримають поведінку переходу, як це встановлено у властивостях transition-duration або transition-timing-function. Якщо потрібно додати кілька властивостей у перехід, вони можуть бути розділені комами всередині значення transition-property. Приклад:

```
 body {
 color: #fff;
 font: 600 14px / 24px "Open Sans", "HelveticaNeue-Light", "Helvetica Neue Light", "Helvetica Neue", Helvetica, Arial, "Lucida Grande", Sans-Serif;
}
.box {
 background: # 2db34a;
 border-radius: 6px;
 cursor: pointer;
 height: 95px;
 line-height: 95px;
 text-align: center;
 transition-property: background, border-radius;
 transition-duration: 1s;
 transition-timing-function: linear;
 width: 95px;
}
.box: hover {
 background: # ff7b29;
 border-radius: 50%;
}
```

Функція переходу

Властивість transition-timing-function використовується для задання швидкості, з якою буде рухатися перехід. Знаючи тривалість з властивості transition-duration, у переходу може бути кілька швидкостей в межах однієї тривалості. Деякі найбільш популярні значення ключових слів для характеристики transition-timing-function включають linear, ease-in, ease-out і ease-in-out.

Значення linear визначає перехід, що рухається з постійною швидкістю з одного стану до іншого. Значення ease-in визначає перехід, який починається повільно і прискорюється протягом переходу, в той час як значення ease-out визначає перехід, який починається швидко і сповільнюється протягом всього переходу. Значення ease-in-out визначає перехід, який починається повільно,

прискорюється в середині, а потім знову сповільнюється перед закінченням. За кожною функцією часу стоїть кубічна крива Безьє, яку можна вказати конкретно за допомогою значення cubic-bezier (x1, y1, x2, y2). Додаткові значення включають step-start, step-stop і значення steps (<число кроків »,« напрямок>). З переходом декількох властивостей ви можете визначити і кілька функцій часу. Ці значення, подібно до інших значень властивостей переходу, можуть бути визначені через кому.

Transition-delay

Крім оголошення властивості переходу, тривалості та функції часу, ви можете також встановити затримку через властивість transition-delay. Затримка задає значення часу, в секундах або мілісекундах, яке визначає, скільки часу перехід має очікувати перед виконанням. Як і з усіма іншими властивостями переходу, щоб затримати численні переходи, кожна затримка може бути перерахована через кому.

```
<Div class = "card-container">
 <Div class = "card">
 <Div class = "side"> <img src = "https://s3-us-west-2.amazonaws.com/s.cdpn.io/29841/jimmy.jpg" alt = "Jimmy Eat World"> < / div>
 <Div class = "side back"> Jimmy Eat World </ div>
 </ Div>
</ Div>
```

```
body {
 font: 600 14px / 24px "Open Sans", "HelveticaNeue-Light", "Helvetica Neue Light", "Helvetica Neue", Helvetica, Arial, "Lu-cida Grande", Sans-Serif;
}
.card-container {
 cursor: pointer;
 height: 150px;
 perspective: 600;
 position: relative;
 width: 150px;
}
.card {
 height: 100%;
 position: absolute;
 transform-style: preserve-3d;
 transition: all 1s ease-in-out;
 width: 100%;
}
.card: hover {
 transform: rotateY (180deg);
}
.card .side {
 backface-visibility: hidden;
```

```
 border-radius: 6px;
 height: 100%;
 position: absolute;
 overflow: hidden;
 width: 100%;
}
.card .back {
 background: #eaeaed;
 color: # 0087cc;
 line-height: 150px;
 text-align: center;
 transform: rotateY (180deg);
}
```

1.4. Правила трансформації CSS

Модель візуального форматування CSS описує систему координат всередині кожного позиціонованого елемента. Система координат є точкою відліку для властивостей зміщення. Положення і розміри в цьому координатному просторі можна розглядати як задані в пікселях щодо точки відліку, з позитивними значеннями, що йдуть вправо і вниз. Цей координатний простір можна змінити за допомогою властивості transform.

CSS3-трансформації дозволяють зрушувати, повертати і масштабувати елементи. Трансформації перетворюють елемент, не зачіпаючи інші елементи веб-сторінки, тобто інші елементи не зрушуються щодо нього.

До елементів, які можуть бути трансформовані, належать елементи з display: block; і display: inline-block ;, а також елементи, значення властивості display яких обчислюється як table-row, table-row-group, table-header-group, table-footer-group, table-cell або table-caption. Трансформованим вважається елемент з будь-яким встановленим значенням властивості transform, відмінним від none.

Існують два види CSS3-трансформацій – **2D** і **3D**.

2D-трансформації перетворюють елементи в двовимірному просторі за допомогою 2D-матриці перетворень. Ця матриця застосовується для обчислення нових координат об’єкта, на основі значень властивостей transform і transform-origin. Перетворення впливають тільки на візуальний рендеринг. Відносно макета сторінки вони можуть відобразитися на переповненні вмісту блоку. За замовчуванням точка трансформації знаходиться в центрі елемента.

Таблиця 1.4 – Властивості трансформацій CSS

Функція	**Опис**
1	2
none	Значення за замовчуванням, означає відсутність трансформації. Також скасовує трансформацію для елемента з групи трансформованих елементів.
matrix (a, c, b, d, x, y)	Зміщує елементи і задає спосіб їх трансформації, дозволяючи об'єднати кілька функцій 2D-трансформацій в одній. У трансформації допустимі поворот, масштабування, нахил і зміна положення. Значення «a» змінює масштаб по горизонталі. Значення від 0 до 1 зменшує елемент, більше 1 – збільшує. Значення «c» деформує (зрушує) сторони елемента по осі Y, позитивне значення – вгору, негативне – вниз. Значення «b» деформує (зрушує) сторони елемента по осі X, позитивне значення – вліво, негативне – вправо. Значення «d» змінює масштаб по вертикалі. Значення менше 1 зменшує елемент, більше 1 – збільшує. Значення «x» зміщує елемент по осі X, позитивне – вправо, негативне – вліво. Значення «y» зміщує елемент по осі Y, позитивне значення – вниз, негативне – вгору.
translate (x, y)	Зрушує елемент на нове місце, переміщуючи відносно звичайного положення вправо і вниз, використовуючи координати X і Y, не зачіпаючи при цьому сусідні елементи. Якщо потрібно зрушити елемент вліво або вгору, то потрібно використовувати негативні значення.
translate3d (x, y, z)	Одночасне переміщення елемента
translateX (n)	Зрушує елемент відносно його звичайного положення по осі X.
translateY (n)	Зрушує елемент відносно його звичайного положення по осі Y.
scale (x, y)	Масштабує елементи, роблячи їх більше або менше. Значення від 0 до 1 зменшують елемент. Перше значення масштабує елемент по ширині, друге – по висоті. Негативні значення відображають елемент дзеркально.
scaleX (n)	Функція масштабує елемент по ширині, роблячи його ширше або вужче. Якщо значення більше 1, елемент стає ширше, якщо значення знаходиться між 1 і 0, елемент стає вужче. Негативні значення відображають елемент дзеркально по горизонталі.

Продовження таблиці 1.4

1	2
scaleY (n)	Функція масштабує елемент по висоті, роблячи його вище або нижче. Якщо значення більше 1, елемент стає вище, якщо значення знаходиться між 1 і 0 – нижче. Негативні значення відображають елемент дзеркально по вертикалі.
scaleZ (x, y)	Масштабує вісь Z.
rotate (кут)	Повертає елементи на задану кількість градусів, негативні значення від –1deg до –360deg повертають елемент проти годинникової стрілки, позитивні – за годинниковою стрілкою. Значення rotate (720deg) повертає елемент на два повних оберти.
skew (x-кут, y-кут)	Використовується для деформування (викривлення) сторін елемента відносно координатних осей. Якщо вказано одне значення, друге буде визначено браузером автоматично.
skewX (кут)	Деформує сторони елемента відносно осі X.
skewY (кут)	Деформує сторони елемента відносно осі Y.
initial	Встановлює значення властивості в значення за замовчуванням.
inherit	Успадковує значення властивості від батьківського елемента.

Жоден стильовий клас не застосовується до першого і останнього <li>. Їхнє становище в ієрархії визначається за допомогою правил CSS.

Якби нам знадобилося додати п'ятий пункт списку, то застосовуючи той самий CSS, стилізація автоматично зміниться.

1.4.1. Використання рухомого рядка під час створення найпростішого анімаційного сюжету

Біжучий рядок створюється за допомогою контейнера тегів ***<marquee> ... </ marquee>*** з такими атрибутами:

– width = "..." – ширина рядка, що біжить в пікселях або відсотках від ширини екрану;

– height = "..." – висота рядка, що біжить в пікселях або відсотках. (Якщо ви робите біжучий рядок в один рядок, то можна висоту не вказувати, вона сама підбирається під розмір літер);

– bgcolor = "..." – визначає колір фону рядка, що біжить;

– behavior = "..." – задає тип руху (поведінку) рядка, що біжить і має такі значення;

– scroll – циклічна прокрутка тексту з одного кінця в інший;

– slide – текст з’являється з одного краю і зупиняється в іншому;

– alternate – текст переміщається від одного краю до іншого і назад;

– direction = "..." – визначає напрямок руху рухомого рядка.

Direction має такі значення:

– left – текст рухається вліво по рядку;

– right – текст рухається вправо по рядку;

– up – весь рядок переміщається від низу до верху;

– down – рядок рухається зверху вниз;

– scrollamount = "..." – крок переміщення в рядку в пікселах, на який переміщується текст за заданий інтервал часу. Наприклад:

– scrollamount = "1" – scrollamount = "2" – scrollamount = "2"

scrolldelay = "..." – цей атрибут задає часовий інтервал між кроками рядка, що біжить в мілісекундах. Наприклад:

– scrolldelay = "100"; – scrolldelay = "200"; – scrolldelay = "300".

Параметр loop = "..." – задає кількість проходів тексту рядка, що біжить. За замовчуванням або із вказівкою значення –1 (infinite) броузер буде прокручувати текст нескінченну кількість разів.

Параметр hspace = "..." – цей атрибут задає поле в пікселах справа і зліва від рядка, що біжить. – hspace = "10" – hspace = "0".

Параметр vspace = "..." – цей атрибут задає відступ в пікселях вище і нижче рядка, що біжить. – hspace = "0" – hspace = "10".

Нижче наведено приклад коду для біжучого рядка розміром 50 пікселів у висоту і 250 пікселів в ширину. Визначена нескінченна прокрутка тексту, текст переміщується зліва направо і має яскраво-блакитне тло.

```
<Marquee loop = "infinite" behavior = "altemate"
 bgcolor = "aqua" direction = "right" height = "50" width = "250">
```

1.5. Контрольні запитання та завдання

1. Яке призначення тегу ?
2. Яке призначення тегу ?
3. Яке призначення тегу ?
4. Поясніть поняття «Hover-ефекту».
5. Як зробити богаторівневе меню сайту?

6. Які класи bootstrap v.3.7. використовують під час побудуви меню сайту?

7. У чому полягає поняття класу CSS, навести приклади?

8. У чому полягає поняття псевдоклас CSS, навести приклади?

9. У чому полягає поняття псевдоелементу CSS, навести приклади?

10. У чому полягає поняття селектору CSS, навести приклади?

11. Які існують засоби побудови адаптованості меню за допомогою бібліотеки Bootstrap?

12. Назвіть правила CSS для створювання медіа-запитів. Призначення медіа-запитів.

13. Назвіть правила CSS для створювання медіа-запитів. Призначення медіа-запитів.

14. Як найти відображену помилку в коді для браузера окрім Explorer для усіх сучасних гаджетів?

15. Як зробити підключення *.CSS файлу у web-сторінку?

16. Чи потрібно було завантажувати проект вашої web-сторінкі у смартфон для остаточної еревірки коду?

17. Вказати призначення тега .

18. Як за допомогою мови Js (java-script) виводити інформацію у вікно браузера?

19. У чому призначення тега <textarea>?

1.6. Завдання для самостійного розв'язання

Створити HTML-сторінку з медіа-запитами, адаптивну для екранів смартфонів і моніторів та настільних пристроїв.

На прикладі наведеної размітки web-сторінки з файлами зображень створити HTML web-сторінку, адаптивну для екранів смартфонів, планшетів і моніторів настільних пристроїв з медіа-запитами.

Для створення медіа-запитів використовується такий синтаксис:

@media умова {

/ * Стилі (вони виконуватимуться, якщо пристрій відповідає зазначеній умові)

}.

Основні типи пристроїв:

• all – усі пристрої (за замовчуванням);

• print – принтери та режим попереднього перегляду сторінки перед друком;

• screen – пристрої з дисплеями;

Логічні оператори:

• and – вимагає обов'язкового виконання всіх зазначених умов.

Наприклад:

@media screen and (min-width: 1200px) and (orientation: landscape) {/ * Стиль CSS}.

Створити html документ такого змісту, який необхідно розташувати в окремий католог, у якому також мають знаходитися спільні бібліотеки jquery-2.2.0.js, bootstrap.min.css, bootstrap.min.js.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title> Приклад використання bootstrap </title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/
bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></s
cript>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js
"></script>
<!-- если не працює інтернет потрібно написати >
<script src=" gruppa/ jq/jquery-2.2.0.js" type="text/javascript"></script>
<link href=" D/студент/gruppa/ /////bootstrap-3.3.6/dist/css/bootstrap.min.css"
rel="stylesheet" media="screen" >
<script src="D/студент/gruppa/////js/bootstrap.min.js" type="text/javascript">
</script>
-->.
```

Побудуйте текстовий файл з розширенням *.html. Для цього необхідно завантажити в нього текст довільного змісту.

Розмістити у каталозі (наприклад) D: \ студент \ група \ Іванов \ images \ шість довільних графічних файлів: 1.jpg, 2.jpg, 3.jpg, 4.jpg, 5.jpg, 6.jpg. Лінійні розміри і розширення у збережених файлах зробити однаковими.

Домогтися відображення в браузері будь-яких чотирьох рисунків, як показано на рис. 1.3, використовуючи наведений нижче код у двох колонках.

```
<div class="image_gallery">
   <div>
      <img src="../images/1.jpg"
title="деревья..." />
```

```
div.image_gallery {
   margin: 0 auto;
   width: 1000px;
   text-align: center;
```

```
    </div>
    <div>
        <img src="../images/2.jpg"
title="…" />
    </div>
    <div>
        <img src="../images/3.jpg"
title="…" />
    </div>
    <div>
        <img src="../images/4.jpg"
title="…" />
    </div>
</div>
```

```
    max-width: 90%; /* контейнер
не превышает 90% ширины
экрана */
    min-width: 500px;
}
img {
    float: left;
    max-width: 48%;
    height: auto;
    padding: 1%; /* небольшие
отступы для изображений */
}
```

Внести зміну до коду, щоб на екрані було зображено шість рисунків з тим самим оформленням стилю, як і в чотирьох на рис. 1.3, але зі зменшеними відступами (настольні ПК – 1100 px):

зображення1	зображення 2	зображення 3
зображення 4	зображення 5	зображення 6

Перевірити у браузері, крім Explorer.

Підібрати медіа-запит, щоб зі зменшенням розміру екрану зображення розташовувалися, як показано на схемі за таблицею (планшети – до 860 px).

зображення 1	зображення 2
зображення 3	зображення 4
зображення 5	зображення 6

Перевірити у браузері, крім Explorer

Підібрати медіа-запит, щоб зі зменшенням розміру екрану зображення розташовувалися, як показано на схемі за таблицею (смартфони – до 768 px).

зображення 1
зображення 2
зображення 3
зображення 4
зображення 5
зображення 6

Перевірити у браузері, крім Explorer

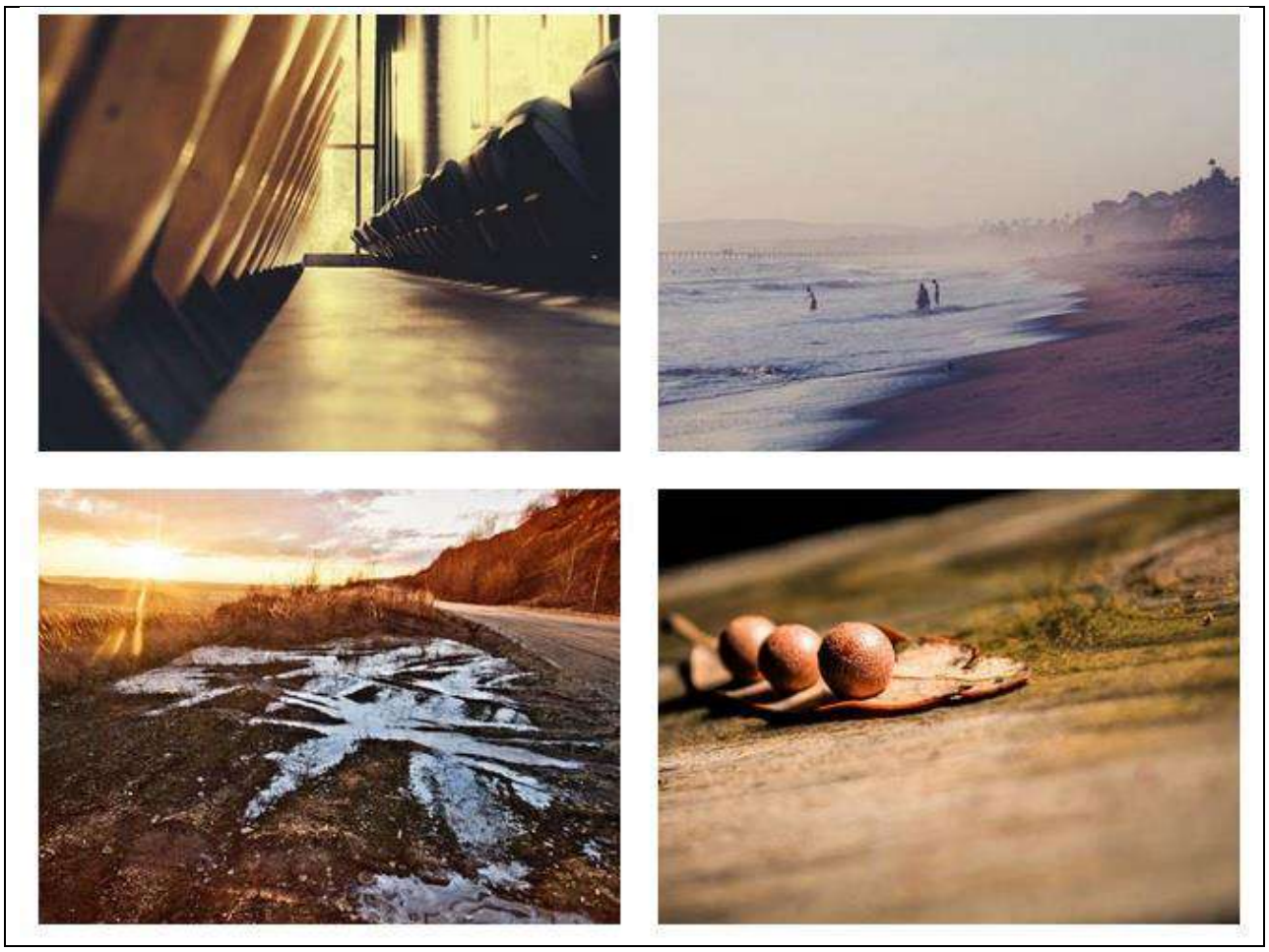


Рисунок 1.3 – Приклад розміщення зображень

Створити окремий файл з назвою Laba2.css з кодом, рекомендований вміст якого наведено в таблиці справа. Підключити створений файл у проект і переконатися у виконанні аналогічних завдань, використовуючи приклади, наведені нижче.

```
/* попередній CSS */
h1 {}
div.image_gallery {}
img {}
@media screen and (max-width:
768px) { /* */
   h1 {
      font-size: 20px;
   }
   div.image_gallery {
      min-width: 320px;
   }
   img {
      max-width: 48%; /* 2 зображеня
в ряд */
      height: auto;
      padding: 1%; /* відступи для
зображень */
   }
```

```
}
@media screen and (max-width:
480px) {    h1 {
        min-width: 320px;
        font-size: 16px;
    }
    div.image_gallery {
        width: 320px;
        min-width: 320px;
    }
    img {
        max-width: 98%;
        height: auto;
        padding: 1%; /*
    }
}
```

```
<link rel="stylesheet" media="screen
and (max-width: 960px)"
href="big.css" />
<link rel="stylesheet" media="screen
and (min-width: 480px)"
href="small.css" />

<!-- для iPhone 4 -->
<link rel="stylesheet" media="only
screen and (-webkit-min-device-pixel-
ratio: 2)" type="text/css"
href="iphone4.css" />

<!-- для iPad portrait and landscape --
>
<link rel="stylesheet" media="all and
(orientation:portrait)"
href="portrait.css">
<link rel="stylesheet" media="all and
(orientation:landscape)"
href="landscape.css">
```

Засоби підключення CSS-таблиць. Так само можна додавати умови, для яких екранів підключатиметься та чи інша таблиця стилів.

Використовуючи правила створення рядка, що біжить, і фрагмент наведеного нижче коду, створити змінне переміщення вгору і вниз довільного зображення з текстом.

Рисунок 1.4 – Приклад зображення, що біжить

Рисунок 1.5 – Приклад зображення, що біжить

```
<div class="boxfix1">
 <div class="layer1" > <img src="img/10.jpg"
alt="img8" class="img-responsive"
id="picture"> <p style="background:
red" >здесь можно разместить
рекламу</p>  </div>
<marquee behavior="alternate"
direction="up"  height="550" vspace="1"
scrolldelay="200"   >
 <img src="img/10.jpg" class="img-
responsive">
<a  style="font-size: 14pt; color:#f25100 "  >
Описание товара   </a> </marquee> </div>
<style>
……...
</style>
```

Рекомендований фрагмент кода

2. ФІЗИЧНІ І МАТЕМАТИЧНІ ОСНОВИ АНІМАЦІЇ

2.1. Криві Безьє

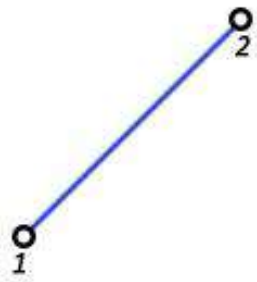
Криві Безьє використовуються в комп'ютерній графіці для малювання плавних вигинів, в CSS-анімації тощо.

Крива Безьє задається опорними точками. Їх може бути дві, три, чотири або більше.

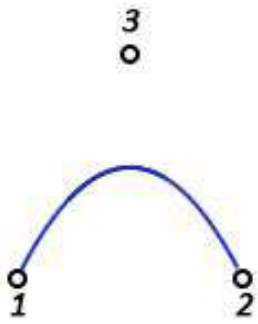
1. **Точки не завжди на кривій.** Це абсолютно нормально. Як саме будується крива, розглянемо пізніше.

2. **Ступінь кривої дорівнює числу точок мінус один.** Для двох точок – це лінійна крива (тобто пряма), для трьох точок – квадратична крива (парабола), для чотирьох – кубічна.

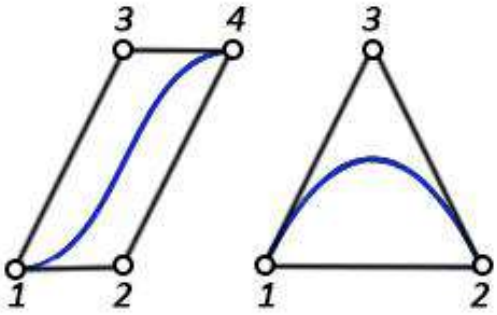
3. **Крива завжди знаходиться всередині** опуклої оболонки**, утвореної опорними точками:**



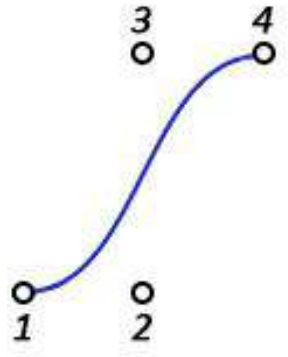
Задання кривої Безьє: за двома точками



Задання кривої Безьє: за трьома точками



Задання кривої Безьє: за чотирма точками



Демонстрація натягнутості кривої Безьє за дотичними 1 → 2 і 3 → 4

Завдяки останній властивості в комп'ютерній графіці можна оптимізувати перевірку перетинів двох кривих. Якщо їх опуклі оболонки не перетинаються, то і криві теж не перетнуться.

Основна цінність кривих Безьє в тому, що, рухаючи точки, криву можна змінювати, причому крива при цьому змінюється інтуїтивно зрозумілим чином.

Як можна помітити, крива натягнута за дотичними 1 → 2 і 3 → 4.

Після невеликої практики стає зрозуміло, як розташувати точки, щоб отримати потрібну форму, а, поєднуючи кілька кривих, можна отримати практично що завгодно. У кривих Безьє є математична формула.

Як ми побачимо далі, для користування кривими Безьє знати формулу немає особливої необхідності, але для повноти картини вона наведена нижче.

Координати кривої описуються в залежності від параметра $t \in [0,1]$.

Для двох точок:

$P = (1–t)\ P1 + tP2$

Для трьох точок:

$P = (1–t)\ 2P1 + 2\ (1–t)\ tP2 + t2P3$

Для чотирьох точок:

$P = (1–t)\ 3P1 + 3\ (1–t)\ 2\ tP2 + 3\ (1–t)\ t2\ P3 + tP4$

Замість Pi потрібно підставити координати *i*-ї опорної точки (xi, yi).

Ці рівняння векторні, тобто для кожної з координат:

$x = (1-t)\ 2x1 + 2\ (1-t)\ tx2 + t2x3$

$y = (1-t)\ 2y1 + 2\ (1-t)\ ty2 + t2y3$

Замість x1, y1, x2, y2, x3, y3 підставляються координати трьох опорних точок, і в той час як t пробігає безліч від 0 до 1, відповідні значення (x, y) саме і утворюють криву.

Втім, це надто наукоподібно, не дуже зрозуміло, чому криві саме такі, і як залежать від опорних точок. З цим нам допоможе розібратися інший, більш наочний алгоритм.

Метод де Кастельжо ідентичний математичному визначенню кривої і наочно показує, як вона будується рис. 2.1.

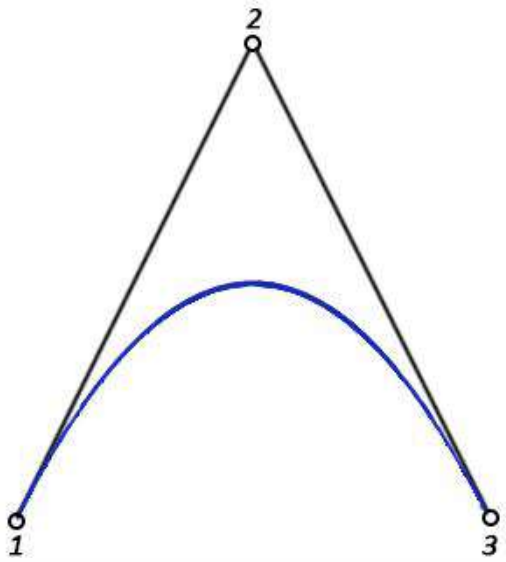


Рисунок 2.1 – Демонстрація за методом де Кастельжо

Алгоритм побудови кривої за «методом де Кастельжо»:

1. Рисуємо опорні точки. В наведеному вище прикладі це 1, 2, 3.

2. Будуємо відрізки між опорними точками 1 → 2 → 3. На рисунку 2.1 вони чорні.

3. Параметр t пробігає значення від 0 до 1. У прикладі вище використаний крок 0.05, тобто в циклі 0,0.05, 0.1, 0.15, ... 0.95, 1.

Значення t пробігає інтервал від 0 до 1, для кожного t:

1. На кожному з цих відрізків береться точка, що знаходиться від початку на відстані від 0 до t пропорційно довжині. Оскільки чорних відрізків два, то і точок виходить дві.

Тобто, при t = 0 точки будуть на початку, при t = 0.25 – на відстані в 25% від початку відрізка, при t = 0.5 – 50% (на середині), при t = 1 – в кінці відрізка.

2. Ці точки з’єднуються. На рис. 2.2 з’єднує їх відрізок зображений синім

3. На отриманому відрізку береться точка на відстані, відповідній t. Тобто, для t = 0.25 (перший рисунок) отримуємо точку в кінці першої чверті відрізка, для t = 0.5 (другий рисунок) – в середині відрізка. На рис. 2.1 та 2.2 ця точка позначена червоним.

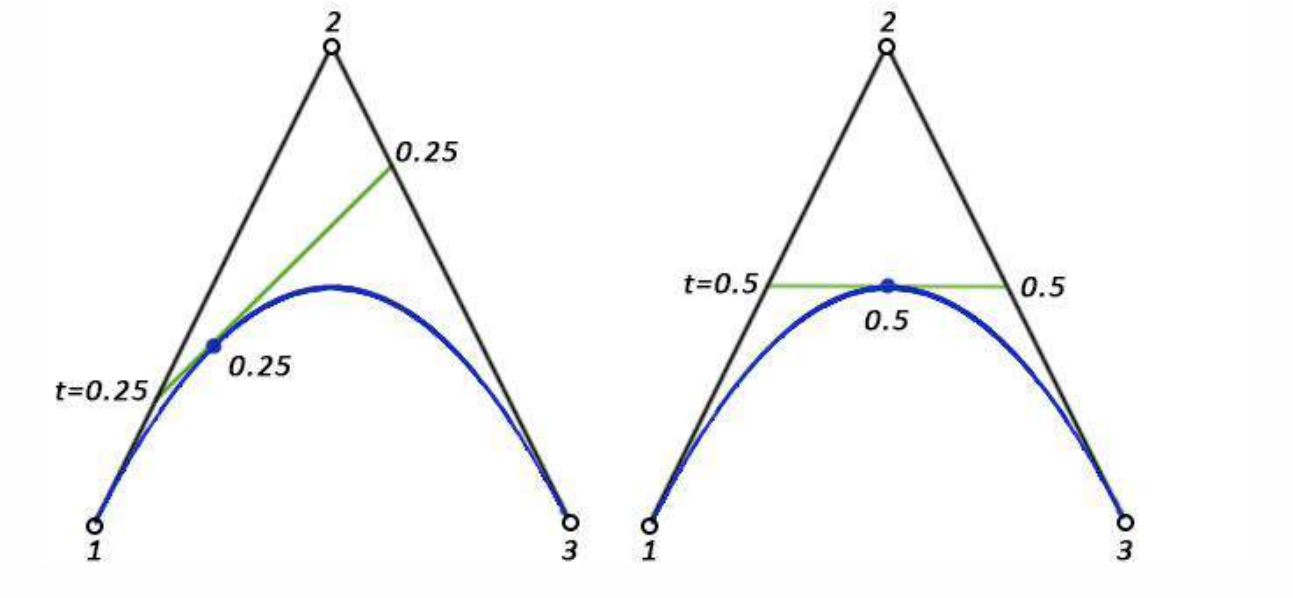


Рисунок 2.2 – Демонстрація переміщення точки на червоній кривій за методом де Кастельжо

4. У міру того як t пробігає послідовність від 0 до 1, кожне значення t додає до кривої точку. Сукупність таких точок для всіх значень t утворюють криву Безьє.

Це був процес для побудови за трьома точками. Але те саме відбувається і з чотирма точками. Як провести криву Безьє через потрібні точки?

Криві Безьє зазвичай проводяться за допомогою «опорних» точок, з яких, як можна бачити з прикладів вище, тільки перша і остання лежать на кривій.

Якщо необхідно провести криву саме через потрібні точки, то це вже інше завдання. Воно називається інтерполяцією. Існують математичні формули, які підбирають коефіцієнти кривої за точками, виходячи з вимог, наприклад, багаточлен Лагранжа.

Як правило, в комп'ютерній графіці для інтерполяції використовують гладко з'єднані кубічні криві. Разом вони виглядають як одна крива. Це називається інтерполяція сплайнами.

Криві Безьє задаються опорними точками.

Ми розглянули два визначення кривих:

– через математичну формулу;

– через процес побудови де Кастельжо.

Їх зручність в тому, що можна легко нарисувати плавні лінії вручну, пересуваючи точки мишкою. Більш складні вигини і лінії можна скласти, якщо з'єднати кілька кривих Безьє в комп'ютерній графіці, моделюванні, у графічних редакторах. Шрифти описуються за допомогою кривих Безьє. У веб-розробці – для графіки на Canvas або у форматі SVG. До речі, всі приклади, показані

вище, написані на SVG. Фактично це один SVG-документ, до якого точки передаються параметрами.

2.2. Контрольні запитання та завдання

1. У чому сутність методу де Кастельжо?
2. У чому призначення кривих Безьє?
3. За скількома точкам будуються криві Безьє?
4. Що являє собою формат SVG?

3. ОПИС МОВИ ПРОГРАМУВАННЯ JAVASCRIPT ДЛЯ ВИКОРИСТАННЯ У РОЗРОБЦІ АНІМАЦІЙНИХ ДОДАТКІВ

3.1. Особливість мови JavaScript

JavaScript був створений програмістом **Brendan Eich** з Netscape і представлений в грудні 1995 року під назвою LiveScript [3, 4]. Досить швидко він був перейменований у JavaScript, хоча офіційною назвою JavaScript є ECMAScript. ECMAScript розробляється і підтримується Міжнародною організацією ECMA (Європейська асоціація виробників комп'ютерів).

Що таке JavaScript?

1. JavaScript – мова сценаріїв, або скриптів. Скрипт являє собою програмний код – набір інструкцій, який не вимагає попередньої обробки (наприклад, компіляції) перед запуском. Код JavaScript інтерпретується движком браузера під час завантаження веб-сторінки. Інтерпретатор браузера виконує порядковий аналіз, обробку і виконання вихідної програми або запиту.

2. JavaScript – об'єктно-орієнтована мова з прототипним спадкуванням. Вона підтримує кілька вбудованих об'єктів, а також дозволяє створювати або видаляти свої власні (призначені для користувача) об'єкти. Об'єкти можуть успадковувати властивості безпосередньо один від одного, утворюючи ланцюжок об'єкт–прототип.

Розглянемо JavaScript на веб-сторінках:

1. Підключення сценаріїв до html-документу

Сценарії JavaScript бувають **вбудовані**, тобто їх вміст є частиною документа, і **зовнішні**, що зберігаються в окремому файлі з розширенням .js. Сценарії можна розмістити у html-документ такими способами:

– у вигляді гіперпосилання.

Для цього потрібно розмістити код в окремому файлі і включити посилання на файл в заголовок. Наприклад, у заголовку:

```
<head>
<script src = "script.js"> </script>
</head>
-або в тіло сторінки:
 <body>
<script src = "D: /users/script.js"> </script>
</body>
```

Цей спосіб зазвичай застосовується для сценаріїв великого розміру або сценаріїв, багаторазово використовуваних на різних веб-сторінках;

– у вигляді обробника події.

Кожен html-елемент має JavaScript-події, які спрацьовують у певний момент. Потрібно додати необхідну подію в html-елемент як атрибут, а як значення цього атрибута вказати потрібну опцію. Функція, що викликається у відповідь на спрацьовування події, є **обробником події.** Внаслідок спрацьовування події виконається пов'язаний з ним код. Цей спосіб застосовується переважно для коротких сценаріїв. Наприклад, можна встановити зміну кольору фону з натисканням на кнопку:

<button onclick = "changeColor ();">	Оголошення кнопки
Змінити колір фону </ button>	
var colorArray = ["# 5A9C6E", "# A8BF5A", "# FAC46E", "# FAD5BB", "# F2FEFF"]; var i = 0; function changeColor ()	Створюємо масив з кольорами фону, змінити колір екрану в циклі
{document.body.style.background = colorArray [i]; i ++; if (i> colorArray.length - 1) {i = 0;	
}}	
body {background: aliceblue}	Стиль екрану за замовчуванням

– всередину елемента<script>

Елемент <script> може вставлятися в будь-яке місце документа. Усередині тега розташовується код, який виконується відразу після прочитання браузером, або містить опис функції, яка виконується в момент її виклику. Опис функції можна розташовувати в будь-якому місці, головне, щоб до моменту її виклику код функції вже був завантажений.

Зазвичай код JavaScript розміщується в заголовку документа (елемент <head>) або після відкриваючого тега <body>. Якщо скрипт використовується після завантаження сторінки, наприклад, код лічильника, то його краще розмістити в кінці документа:

```
<footer>
<script>
document.write ( "Введіть своє ім'я");
</ script>
</ footer>
</ body>
```

3.1.1. Типи даних і змінні в JavaScript

Комп'ютери обробляють інформацію – дані. Дані можуть бути наведені в різних формах або типах. Велика частина функціональності JavaScript реалізується за рахунок простого набору об'єктів і типів даних. Функціональні можливості, пов'язані з рядками, числами і логікою, базуються на рядкових, числових і логічних типах даних. Інша функціональна можливість, що містить регулярні вирази, дати і математичні операції, здійснюється за допомогою об'єктів RegExp, Date і Math.

Літерали в JavaScript є особливим класом типу даних, фіксовані значення одного з трьох типів даних – рядкового, числового чи логічного:

```
"це рядок"
3.14
true
alert ( "Hellow"); // "Hellow" - це літерал
var myVariable = 15; // 15 - це літерал
```

Примітивний тип даних є екземпляром певного типу даних, таких як рядковий, числовий, логічний, null і undefined.

Дані, що обробляються сценарієм JavaScript, є **змінними**. Змінні являють собою іменовані контейнери, що зберігають дані (значення) в пам'яті комп'ютера, які можуть змінюватися в процесі виконання програми. Змінні мають **ім'я, тип** та **значення.**

Ім'я змінної, або **ідентифікатор,** може містити тільки літери a–z, A–Z, цифри 0–9 (цифра не може бути першою в імені змінної), символ $ (може бути тільки першим символом в імені змінної або функції) і символ підкреслення, наявність прогалин не допускається. Довжина імені змінної не обмежена. Можна, але не рекомендується, записувати імена змінних літерами українського алфавіту, для цього вони мають бути записані в Unicode.

Як ім'я змінної не можна використовувати ключові слова JavaScript. Імена змінних в JavaScript чутливі до регістру, що означає, що змінна var message; і var Message; – різні змінні.

Змінна створюється (оголошується) за допомогою ключового слова var, за яким йде ім'я змінної, наприклад, var message ;. Оголошувати змінну необхідно перед її використанням.

Змінна ініціюється значенням за допомогою операції привласнення =, наприклад, var message = "Hellow" ;, тобто створюється змінна message і в ній зберігається її початкове значення "Hellow". Змінну можна оголошувати без значення, в цьому випадку їй присвоюється значення за замовчуванням

undefined. Значення змінної може змінюватися під час виконання скрипта. Різні змінні можна оголошувати в одному рядку, розділивши їх комою.

3.2. DOM-модель веб-сайту

Основним інструментом роботи і динамічних змін на сторінці є DOM (DocumentObject Model) – об'єктна модель, використовувана для XML / HTML-документів. Згідно з DOM-моделлю, документ є ієрархією, деревом. Кожен HTML-тег утворює вузол дерева з типом «елемент». Вкладені в нього теги стають дочірніми вузлами. Для подання тексту створюються вузли з типом «текст».

DOM – це уявлення документа у вигляді дерева об'єктів, доступне для зміни через JavaScript.

Побудуємо, для початку, дерево DOM для такого документа.

У цьому дереві виділено два типи вузлів.

1. Теги утворюють вузли-елементи (element node). Природним чином одні вузли вкладені в інші. Структура дерева утворена виключно за рахунок них.

Приклад DOM:

```
<! DOCTYPE HTML>
<html>
<head>
 <title> Про лосів </ title>
</ head>
<body>
 Правда про лосів
</ body>
</ html>
```

2. Текст всередині елементів утворює текстові вузли (text node), позначені як #text.

Текстовий вузол містить виключно рядок тексту і не може мати нащадків, тобто він завжди на найнижчому рівні.

Зверніть увагу на спеціальні символи в текстових вузлах:

Навіщо, крім зображень, потрібна ієрархічна модель DOM?

DOM потрібен для того, щоб маніпулювати сторінкою – читати інформацію з HTML, створювати і змінювати елементи.

Вузол HTML можна отримати як document.documentElement, а BODY – як document.body. Отримавши вузол, ми можемо щось зробити з ним.

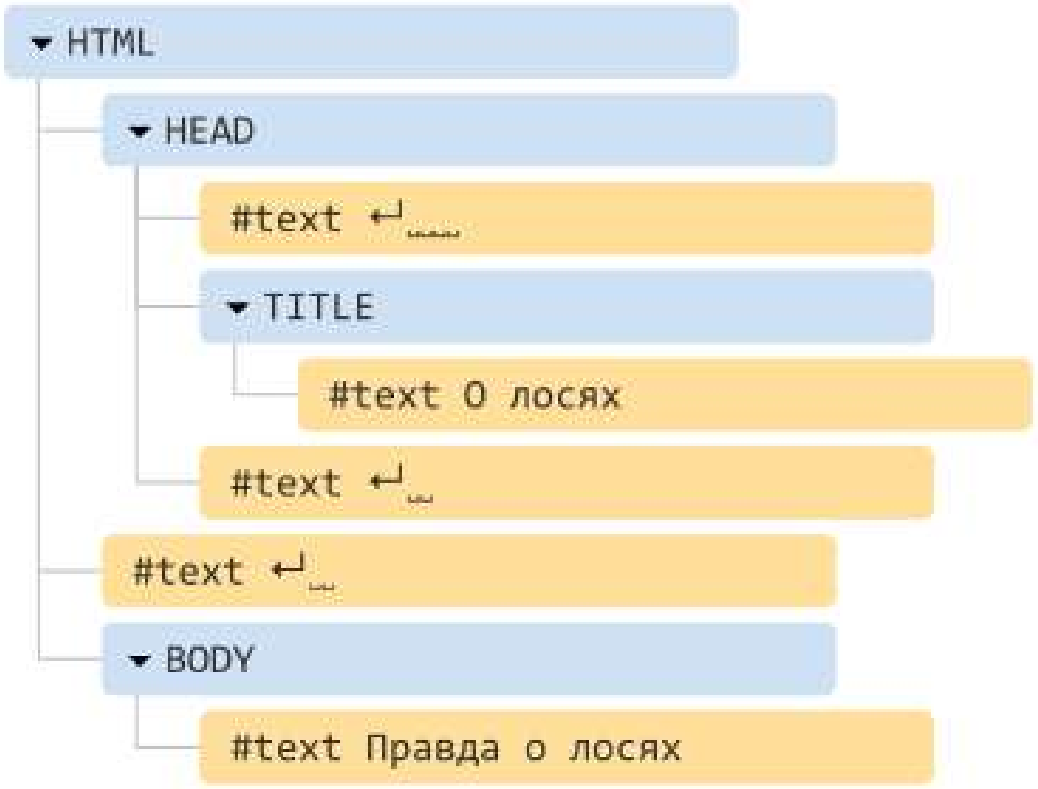


Рисунок 3.1 – Демонстрація DOM-моделі

Наприклад, можна поміняти колір BODY і повернути назад:
DOM надає можливість робити зі сторінкою все, що завгодно.
Розглянемо різні властивості і методи DOM-вузлів.

```
document.body.style.backgroundColor = 'red';
alert ( 'Поміняли колір BODY');
document.body.style.backgroundColor = '';
alert ( 'Скинули колір BODY');
```

Так виглядають основні посилання, за якими можна переходити між вузлами DOM: верхні елементи дерева доступні безпосередньо з document.

<*HTML*> = *document.documentElement*

Перша точка входу – document.documentElement. Це властивість посилається на DOM-об’єкт для тега <html>.

DOM дозволяє робити що завгодно з HTML-елементом і його вмістом, але для цього необхідно спочатку отримати потрібний елемент.

Доступ до DOM починається з об’єкта document. З нього можна дістатися до будь-яких вузлів.

Дочірні елементи (або діти) – елементи, які лежать безпосередньо всередині даного (childNodes, firstChild, lastChild).

Наприклад, всередині <HTML> зазвичай лежать <HEAD> і <BODY>.

Нащадки – всі елементи, які лежать всередині даного, разом з їхніми дітьми, дітьми їхніх дітей і так далі. Тобто, все піддерево DOM.

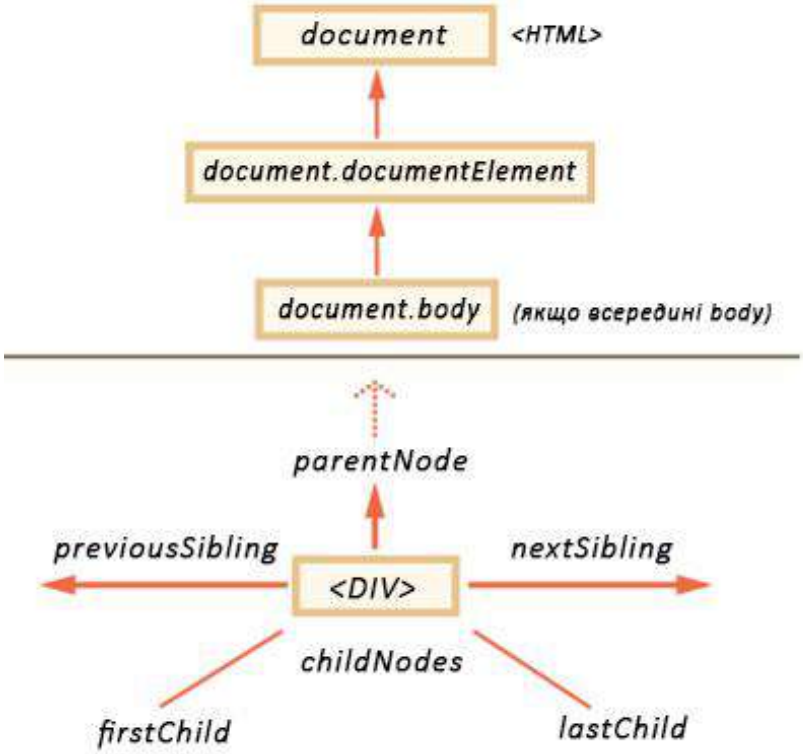


Рисунок 3.2 – Структура DOM

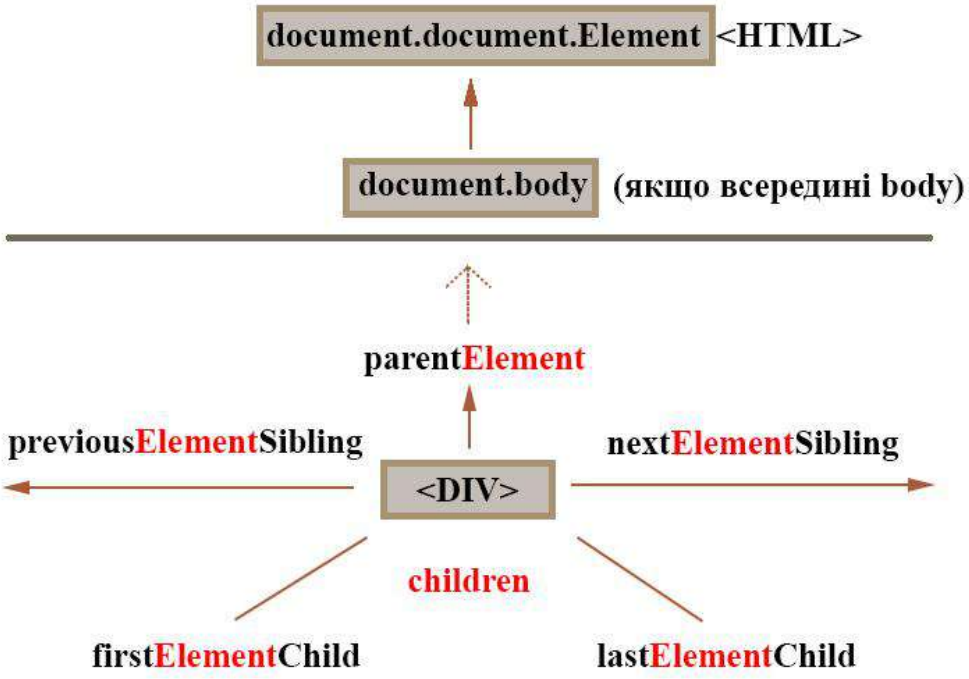


Рисунок 3.3 – Ієрархія DOM моделі

Children – тільки дочірні вузли-елементи, тобто відповідні тегам.

FirstElementChild, lastElementChild – відповідно, перший і останній діти-елементи.

PreviousElementSibling, nextElementSibling – сусіди-елементи.

ParentElement – батько-елемент.

```
<! DOCTYPE HTML>
<html>
<body>
 <div> Початок </ div>
 <ul>
 <li> Інформація </ li>
 </ ul>
 <div> Кінець </ div>
</ body>
</ html>
<script>
 for (var i = 0; i <document.body.childNodes.length; i ++) {
 alert (document.body.childNodes [i]);
// Text, DIV, Text, UL, ..., SCRIPT
 }
 </ script>
```

Приклад нижче послідовно виведе дочірні елементи document.body

```
<body>
 <div> Читачі: </ div>
 <ul>
 <li> Вася </ li>
 <li> Петя </ li>
 </ ul>
 <!- коментар ->
 </ body>
<script>
 var childNodes = document.body.childNodes;
 for (var i = 0; i <childNodes.length; i ++) {
 // відфільтрувати не-елементи
 if (childNodes [i] .nodeType! = 1) continue;
 alert (childNodes [i]);
 }
 </ script>
```

Наприклад, виведемо всі вузли-нащадки document.body, кі є елементами: тип вузла можна тільки читати, змінити його неможливо. Тип: nodeType

Нижче наведена таблиця методів доступу до елементів сайту.

Таблиця 3.1 – Методи навігації по сайту

Метод	Шукає за ...	Підтримка
getElementById	id	IE, OPERA, CHROME MOZILLA, FIREFOX
getElementsByName	ім’я	скрізь
getElementsByTagName	тегу	скрізь
getElementsByClassName	класу	скрізь
querySelector CSS	селектор	скрізь
querySelectorAll	CSS-селектор	скрізь

Вузли DOM також мають інші властивості, залежно від тега. Наприклад, у INPUT є властивості value і checked, а у A є href і т.д. Ми розглянемо їх далі.

Основні властивості DOM-вузлів:

– nodeType

Тип вузла. Найпопулярніші типи: «1» – для елементів і «3» – для текстових вузлів. Тільки для читання;

– nodeName / tagName

Назва тега великими літерами. nodeName має спеціальні значення для вузлів-неелементів. Тільки для читання;

– innerHTML

Внутрішній вміст сайту-елемента у вигляді HTML. Можна змінювати;

– outerHTML

Повний HTML вузла-елемента. Під час запису в elem.outerHTML змінна elem зберігає старий вузол;

– nodeValue/data

Вміст текстового вузла або коментаря. Властивість nodeValue також визначена і для інших типів вузлів. Можна змінювати.

Вузли DOM також мають інші властивості, залежно від тега. Наприклад, у INPUT є властивості value і checked, а у A є href і т.д.

Створення і видалення DOM елементів

Вузол DOM – це об’єкт, тому, як і будь-який об’єкт в JavaScript, він може містити призначені для користувача властивості і методи.

Зміна DOM – ключ до створення «живих» сторінок.

Як приклад розглянемо додавання повідомлення на сторінку, щоб воно було оформлене красивіше ніж звичайний alert.

*** ***
*** Ура! </ Strong> Ви прочитали це важливе повідомлення.***

```
</ div>
<style>
.alert {
 padding: 15px;
 border: 1px solid # d6e9c6;
 border-radius: 4px;
 color: # 3c763d;
 background-color: # dff0d8;
}
</style>
```

Для створення елементів використовуються описані нижче методи.

Створює новий елемент із зазначеним тегом:

document.createElement (tag),
var div = document.createElement ('div').

Створює новий текстовий вузол з даним текстом:

document.createTextNode (text),
var textElem = document.createTextNode ('Тут був я ").

Розглянемо приклад створення повідомлення.

У нашому випадку ми хочемо зробити DOM-елемент div, дати йому класи і заповнити текстом:

```
var div = document.createElement ('div');
div.className = "alert alert-success ";
div.innerHTML = "<strong> Ура! </ strong> Ви прочитали це важливе повідомлення."
```

Після цього коду у нас є готовий DOM-елемент. Поки що він присвоєний в змінну div, але його не видно, адже він не пов’язаний зі сторінкою.

Щоб DOM-вузол був показаний на сторінці, його необхідно вставити в document. Для цього насамперед потрібно вирішити, куди ми будемо його вставляти. Припустимо, ми вирішили, що вставляти будемо в якийсь елемент parentElem, наприклад var parentElem = document.body. Для вставки всередину parentElem є такі методи: parentElem.appendChild (elem).

«Append» – додає elem в кінець дочірніх елементів parentElem.

Додавання елемента:

appendChild, insertBefore, parentElem.insertBefore (elem, nextSibling)

Вставляє elem в колекцію дітей parentElem, перед елементом nextSibling.

```
<ol id = "list">
 <li> 0 </ li>
 <li> 1 </ li>
 <li> 2 </ li>
</ ol>
```

```
<script>
 var newLi = document.createElement ( 'li');
 newLi.innerHTML = 'Привіт, світ!';
 list.appendChild (newLi);
</script>
```

Наступний код вставляє новий елемент попереду другого <li>:

Для вставки елемента в початок досить вказати, що вставляти будемо перед першим нащадком тому, що якщо другим аргументом вказати null, то insertBefore спрацює як метод appendChild.

Так що метод insertBefore універсальний, що показано на прикладі нижче.

```
<ol id = "list">
 <li> 0 </ li>
 <li> 1 </ li>
 <li> 2 </ li>
</ ol>
```

```
<script>
 var newLi = document.createElement ('li');
 newLi.innerHTML = 'Привіт, світ!';
 list.insertBefore (newLi, list.children [1]);
</script>
```

Події

Приклад події на основі завдання з анімацією м’яча наведений на рис. 3.4.

Рисунок 3.4 –Демонстрація анімації з використанням JS і HTML/CSS

Завдання: зробіть так, щоб м’яч (рис. 3.4) підстрибував.

В HTML/CSS, падіння м'яча можна відобразити зміною властивості ball.style.top від 0 і до значення, відповідного нижньому положенню.

Нижня межа елемента field, в якому знаходиться м’яч, має значення field.clientHeight. Але властивість top належить до верху м’яча, тому вона змінюється до field.clientHeight- ball.clientHeight.

Для створення анімаційного ефекту найкраще підійде функція bounce в режимі easeOut.

Для анімування м’яча створюється підстрибуючий м’яч. Нам потрібно лише додати ще одну анімацію для elem.style.left.

Горизонтальна координата змінюється за іншим законом, ніж вертикальна. Вона не «підстрибує», а постійно збільшується, поступово зрушуючи м’яч вправо.

Тому ми не можемо додати її в той самий animate, потрібно робити окремий. Як часову функцію для переміщення вправо ми могли б застосувати для неї linear, але тоді горизонтальний рух буде відставати від стрибків м’яча. Більш красиво буде щось типу makeEaseOut (quad).

```
var height = field.clientHeight - ball.clientHeight;
var width = 100;
animate ({
 duration: 2000,
 timing: makeEaseOut (bounce),
 draw: function (progress) {
 ball.style.top = height * progress + 'px'
 }
});
animate ({
 duration: 2000,
 timing: makeEaseOut (quad),
 draw: function (progress) {
 ball.style.left = width * progress + "px"
 }
});

<! DOCTYPE HTML>
<html>

<head>
 <style>
 #field {
 width: 200px;
 border: 10px groove black;
 background-color: # 00FF00;
 position: relative;
 }

 #ball {
 position: absolute;
 }
 </ style>
</ head>
```

```
<body>
 <div id = "field">
<! - імпорт зображення з м'ячем, можна використовувати малюнок схожого змісту і відповідний графічний файл довільного формату>
 <img src = "https://en.js.cx/clipart/ball.svg" width = "40" height = "40" id = "ball">.
 </div>
 <script>
 // ball.offsetWidth = 0 до того, як зображення завантажилося!
 // виправимо це, встановивши ширину:
 ball.style.left = Math.round (field.clientWidth / 2 - ball.offsetWidth / 2) + 'px'
 ball.style.top = Math.round (field.clientHeight / 2 - ball.offsetHeight / 2) + 'px'
 </script>
</body>
</html>
```

Обчисліть їх і використовуйте, щоб помістити м’яч в центр поля:

– елемент має позиціонуватися за рахунок JavaScript, а не CSS;

– код має працювати з будь-яким розміром м’яча (10, 20, 30 пікселів) і будь-яким розміром поля без прив’язки до початкових значень.

Центрування можна зробити за допомогою тільки CSS, але завдання саме на JavaScript. Далі будуть інші теми і більш складні ситуації, коли JavaScript буде вже точно необхідний. Це свого роду «розминка».

Розглянемо вирішення задачі.

М’яч має CSS-властивість position: absolute. Це означає, що координати left/top вимірюються щодо найближчого позиційованого елемента, яким є #field (оскільки у нього є CSS властивість position: relative).

Координати відраховуються від внутрішнього верхнього лівого кута поля.

Ширина і висота внутрішнього поля – це clientWidth/clientHeight. Таким чином, його центр має координати (clientWidth/2, clientHeight/2). Але якщо ми встановимо м’ячу такі значення ball.style.left/top, то в центрі буде не сам м’яч, а його лівий верхній кут:

```
ball.style.left = Math.round (field.clientWidth / 2) + 'px';
ball.style.top = Math.round (field.clientHeight / 2) + 'px';
```

Для того, щоб центр м’яча знаходився в центрі поля, нам потрібно змістити м’яч на половину його ширини вліво і на половину його висоти вгору:

```
ball.style.left = Math.round (field.clientWidth / 2) + 'px';
ball.style.top = Math.round (field.clientHeight / 2) + 'px';
```

Код вище стабільно працювати не буде, тому що <img> йде без ширини/висоти:

<img src = "ball.png" id = "ball">

Якщо браузеру невідомі ширина і висота зображення (з атрибута HTML-тега або CSS-властивостей), він вважає їх рівними 0 до тих пір, поки зображення не завантажиться.

При першому завантаженні браузер зазвичай кешує зображення, так що під час наступного завантаження воно буде доступно відразу, разом з розмірами. Але при першому завантаженні значення ширини м'яча ball.offsetWidth дорівнює 0. Це призводить до обчислення невірних координат.

Ми можемо виправити це, додавши атрибути width / height тегу <img>:

```
<img src = "ball.png" width = "40" height = "40" id = "ball">
...Або задавши розміри в CSS:
#ball {
 width: 40px;
 height: 40px;
}
```

У наступному варіанті вирішення задачі використовується спеціалізована бібліотека animate.js.

```
<! DOCTYPE HTML>
<html>

<head>
 <script src = "https://js.cx/libs/animate.js"> </ script>
 <link rel = "stylesheet" href = "style.css">
</ head>

<body>

 <div id = "field">
 <img src = "https://js.cx/clipart/ball.svg" width = "40" height = "40" id = "ball">
 </ div>

 <script>
 function makeEaseOut (timing) {
 return function (timeFraction) {
 return 1 - timing (1 - timeFraction);
 }
 }
```

```
function bounce (timeFraction) {
for (let a = 0, b = 1, result; 1; a + = b, b / = 2) {
if (timeFraction> = (7 - 4 * a) / 11) {
return -Math.pow ((11 - 6 * a - 11 * timeFraction) / 4, 2) + Math.pow (b, 2)
}
}
}
ball.onclick = function () {
let to = field.clientHeight - ball.clientHeight;
animate ({
duration: 2000,
timing: makeEaseOut (bounce),
draw (progress) {
ball.style.top = to * progress + 'px'
}
});
};
</ script>
</ body>
</ html>
#field {
height: 200px;
border-bottom: 3px black groove;
position: relative;
}
#ball {
position: absolute;
cursor: pointer;
}
```

3.3. Взаємодії мови програмування JavaScript з елементами розмітки і стилями (CSS)

3.3.1. Введення в браузерні події

Подія – це сигнал від браузера про те, що щось сталося. Для реакції події користувача і внутрішньої взаємодії скриптів існують події. Існує багато видів подій. Розглянемо список найчастіше використовуваних, поки просто для ознайомлення:

Події мишки:

click – відбувається, коли клацали на елемент лівою кнопкою мишки;

contextmenu – відбувається, коли клацали на елемент правою кнопкою мишки;

mouseover – виникає, коли на елемент наводиться мишка;
mousedown і ***mouseup*** – коли кнопку мишки натиснули або відпустили;
mousemove – із рухом мишки.

Події на елементах управління:
submit – відвідувач відправив форму <form>;
focus – відвідувач фокусується на елементі, наприклад натискає на <input>;

Клавіатурні події:
keydown – коли відвідувач натискає клавішу;
keyup – коли відвідувач відпускає клавішу.

Події документа:

DOMContentLoaded – коли HTML завантажений і оброблений, DOM-документ повністю побудований.

Події можна призначити обробник, тобто функцію, яка спрацює, як тільки подія відбулась. Лише завдяки обробникам JavaScript-код може реагувати на дії користувача. Є кілька способів призначити події обробника. Ми їх розглянемо, починаючи від самого простого.

Використання атрибута HTML. Оброблювач може бути призначений прямо в розмітці, в атрибуті, який називається on <подія>. Наприклад, щоб прикріпити click-подію до input кнопці, можна привласнити обробник onclick.

З натисканням мишкою на кнопці виконається код, вказаний в атрибуті onclick. Зверніть увагу, для вмісту атрибута onclick використовуються одинарні лапки, оскільки сам атрибут знаходиться в подвійних.

Часта помилка в тому, що забувається, що код знаходиться всередині атрибута. Запис вигляду onclick = «alert("Клік!")Я", з подвійними лапками всередині, не працюватиме. Якщо вам дійсно потрібно використовувати саме подвійні лапки, то це можна зробити, замінивши їх на & quot ;, тобто так: onclick = "alert (& quot; Клік! & Quot;)".

Втім зазвичай цього не потрібно, оскільки прямо в розмітці пишуться тільки дуже прості обробники. Якщо потрібно зробити щось складне, то має сенс описати це в функції, і в обробнику викликати вже її.

Наступний приклад з натисканням запускає функцію countRabbits ().

Приклад призначення обробника подій:

<Input value = "Натисни мене" onclick = "alert ('Клік!')" Type = "button">.

Атрибут HTML-тега не чутливий до регістру, тому ONCLICK працюватиме так само, як onClick або onCLICK... Але, як правило, атрибути пишуть в нижньому регістрі: onclick.

Приклад розгляду використання властивості DOM-об'єкта. Можна призначати обробник, використовуючи властивість DOM-елемента on <подія>.

Якщо обробник заданий через атрибут, то браузер читає HTML-розмітку, створює нову функцію з вмісту атрибута і записує у властивість onclick. Цей спосіб, по суті, аналогічний попередньому. Оброблювач зберігається саме в DOM-властивості, а атрибут – лише один із способів його ініціалізації.

Ці два приклади коду працюють однаково:

1) тільки HTML;

2) HTML + JS.

Приклад 1. Тільки HTML

```
<! DOCTYPE HTML>
<html>
<head>
 <meta charset = "utf-8 ">
</ head>
<body>
 <input type = "button" onclick = "countRabbits ()" value = "Вважати кроликів!" />
</ body>
</ html>
<script>
 function countRabbits () {
 for (var i = 1; i <= 3; i ++) {
 alert ( "Кролик номер" + i);
 }
 }
 </ script>
```

Приклад 2. HTML + JS

```
<input id = "elem" type = "button" value = "Натисни мене" />
<script>
 elem.onclick = function () {
 alert ( 'Дякую');
 };
</ script>
<input type ="button" onclick ="alert('Клік!')" value = "Кнопка"/>.
```

Оскільки DOM-властивість onclick, в результаті, одне, то призначити більше одного обробника так не можна. У прикладі нижче призначення через JavaScript перезапише обробник з атрибута:

До речі, оброблювачем можна призначити і вже існуючу функцію.

Якщо обробник перестав бути потрібним – його завжди можна прибрати призначенням elem.onclick = null.

Усередині обробника події this посилається на поточний елемент, тобто на той, на якому він спрацював.

Це можна використовувати, щоб отримати властивості або змінити елемент.

У коді нижче button виводить свій вміст, використовуючи this.innerHTML.

Зверніть увагу на такі особливості:

– функція має бути присвоєна як sayThanks, а не sayThanks ().

```
<input type = "button" id = "button" value = "Кнопка" />
<script>
 button.onclick = function () {
 alert ( 'Клік!');
 };
</ script>
<input type = "button" id = "elem" onclick = "alert ( 'До')" value = "Натисни мене" />
<script>
 elem.onclick = function () {// перезапише існуючий обробник
alert ( 'Після'); // виведеться тільки це
 };
</script>
Натисни мене
function sayThanks () {
 alert ( 'Дякую!');
}
elem.onclick = sayThanks;
```

Доступ до елемента через this

```
<Button onclick = "alert (this.innerHTML)"> Натисни мене </ button>.
```

Якщо додати дужки, то sayThanks () – буде вже результат виконання функції (а оскільки в ній немає return, то в onclick потрапить undefined).

Нам потрібна саме функція. А ось в розмітці саме дужки потрібні.

Цю різниця просто пояснити. Зі створенням обробника браузером з атрибута, він автоматично створює функцію з його вмісту.

Призначення обробника рядком elem.onclick = "alert (1)" можна іноді побачити в застарілому коді. Це працюватиме, але не рекомендується, можуть бути проблеми зі стисненням JavaScript. Та й взагалі, передавати код у вигляді рядка щонайменше дивно в мові, яка підтримує Function Expressions. Це можливо лише з міркувань сумісності, не робіть так. Не використовуйте setAttribute.

Фундаментальний недолік описаних вище способів призначення обробника – неможливість повісити кілька обробників на одну подію.

Наприклад, одна частина коду хоче із натисканням на кнопку робити її підсвічуваною, а інша – видавати повідомлення. Потрібно в різних місцях два обробника повісити. При цьому новий обробник затиратиме попередній. Наприклад, наступний код насправді призначає один обробник – останній:

```
button.onclick = sayThanks;
<input type = "button" id = "button" onclick = "sayThanks ()" />
button.onclick = function () {
sayThanks (); // вміст атрибута
};
// з натисканням на body будуть помилки
// оскільки з призначенням в атрибут функція буде перетворена в рядок
document.body.setAttribute ( 'onclick', function () {alert (1)}).
```

Недолік призначення через властивість очевидний. Розробники стандартів досить давно це зрозуміли і запропонували альтернативний спосіб призначення обробників за допомогою спеціальних методів, які вільні від зазначеного недоліку.

Методи addEventListener і removeEventListener є сучасним способом призначити або видалити обробник, і при цьому дозволяють використовувати скільки завгодно будь-яких оброблювачів.

Призначення обробника здійснюється викликом addEventListener з трьома аргументами: event.

Ім’я події, наприклад click handler.

Видалення обробника здійснюється викликом removeEventListener:

```
input.onclick = function () {alert (1); }
// ...
input.onclick = function () {alert (2); } // замінить попередній обробник
addEventListener і removeEventListener
element.addEventListener (event, handler [, phase]);
// передати ті ж аргументи, що були у addEventListener
element.removeEventListener (event, handler [, phase]);
```

Для видалення потрібно передати саме ту функцію-обробник, яка була призначена. Ось так removeEventListener не спрацює.

У removeEventListener передана не та сама функція, а інша, з однаковим кодом, але це не важливо.

Звернемо увагу: якщо функцію не зберегти де-небудь, а просто передати в addEventListener, як у попередньому коді, то потім отримати її назад, щоб зняти обробник, буде неможливо. Немає методу, який дозволяє враховувати обробники подій, призначені через addEventListener.

Метод addEventListener дозволяє додавати кілька обробників на одну подію одного елемента.

Як видно з прикладу вище, можна водночас призначати обробники і через DOM-властивість і через addEventListener. Проте, щоб уникнути плутанини, рекомендується вибрати один спосіб.

```
elem.addEventListener ( "click", function () {alert ( 'Дякую!')});
// ....
elem.removeEventListener ( "click", function () {alert ( 'Дякую!')});
function handler () {
 alert ( 'Дякую!');
}
input.addEventListener ( "click", handler);
// ....
input.removeEventListener ( "click", handler);
<input id = "elem" type = "button" value = "Натисни мене" />
<script>
 function handler1 () {
 alert ( 'Дякую!');
 };
 function handler2 () {
 alert ( 'Спасибі ще раз!');
 }
elem.onclick = function () {alert ( "Привіт"); };
elem.addEventListener ( "click", handler1); // Дякуємо!
elem.addEventListener ( "click", handler2); // Дякую ще раз!
</script> addEventListener працює завжди, а DOM-властивість – немає.
```

У спеціальних методах є ще одна перевага перед DOM-властивостями. Є деякі події, які не можна призначити через DOM-властивість, але можна через addEventListener. Наприклад, подія transitionend, тобто закінчення CSS-анімації. У більшості браузерів вона вимагає призначення через

addEventListener. Ви можете перевірити це, запустивши код у прикладі нижче. Як правило, спрацює лише другий обробник, але не перший.

У IE8 – замість addEventListener/removeEventListener використовуються свої методи, призначення обробника здійснюється викликом attachEvent.

Видалення обробника – викликом detachEvent:

```
<button id = "elem" onclick = "this.classList.toggle ( 'wide');">
 Натисни мене
</button>
<style>
 button {
 transition: width 1s;
 width: 100px;
 }
 .wide {
 width: 300px;
 }
</style>
<script>
elem.ontransitionend = function () {alert ( "ontransitionend"); // не спрацює
 };
 elem.addEventListener ( "transitionend", function () {
alert ( "addEventListener"); // спрацює після закінчення анімації
 });
</script>
```

Відмінності IE8 – element.attachEvent («on» + event, handler).

Як бачите, майже те саме, тільки подія має включати префікс on.

У оброблювачів, призначених з attachEvent, немає this (слово this означає прив’язку до об’єкта).

Є три способи призначення обробників подій:

1. Атрибут HTML: onclick = "...".

2. Властивість: elem.onclick = function.

3. Спеціальні методи:

Сучасні: elem.addEventListener (подія, handler [, phase]), видалення через removeEventListener.

Для старих IE8-: elem.attachEvent (on + подія, handler), видалення через detachEvent. Порівняння addEventListener і onclick:

```
element.detachEvent ( "on" + event, handler);
function handler () {
 alert ( 'Дякую!');
}
button.attachEvent ( "onclick", handler) // Призначення обробника
 // ....
button.detachEvent ( "onclick", handler) // Видалення обробника
```

Деякі події можна призначити тільки через addEventListener. Метод addEventListener дозволяє призначити багато оброблювачів на одну подію. Оброблювач, призначений через onclick, простіше видалити або замінити.

Метод onclick крос-браузерний. Цим введенням ми тільки відкриваємо роботу з подіями, але ви вже можете вирішувати різноманітні завдання з їх використанням.

3.3.2. Поняття «спливання і занурення»

Давайте відразу почнемо з прикладу.

Цей оброблювач для спрацює, якщо ви натиснете за вкладеним тегом або <code>:

```
<div onclick = "alert ( 'Оброблювач для Div спрацював!')">
 <em> Натисніть на <code> EM </ code>, спрацює обробник на <code> DIV
</ code> </ em>
</ div>
```

Натисніть на EM, спрацює обробник на DIV. З настанням події обробники спочатку спрацьовують на самому вкладеному елементі, потім на його батьківському елементі, потім вище і так далі, вгору за ланцюжком вкладеності.

```
<form onclick = "alert ( 'form')"> FORM
<div onclick = "alert ( 'div')"> DIV
<p onclick = "alert ( 'p')"> P </ p>
</ div>
</ form>
<style>
 body * {
 margin: 10px;
 border: 1px solid blue;
 }
</ Style>
```

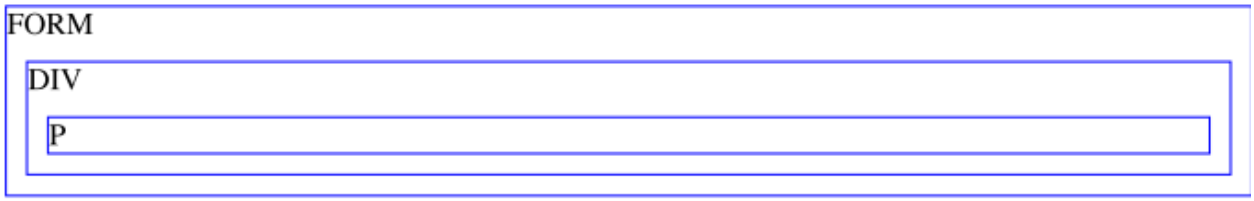


Рисунок 3.5 – Демонстрація вкладеності елементів розмітки

Спливання гарантує, що натискання по внутрішньому <p> викличе обробник onclick (якщо є) спочатку на самому <p>, потім на елементі далі на елементі <form>, і так далі вгору за ланцюжком батьків до самого document.

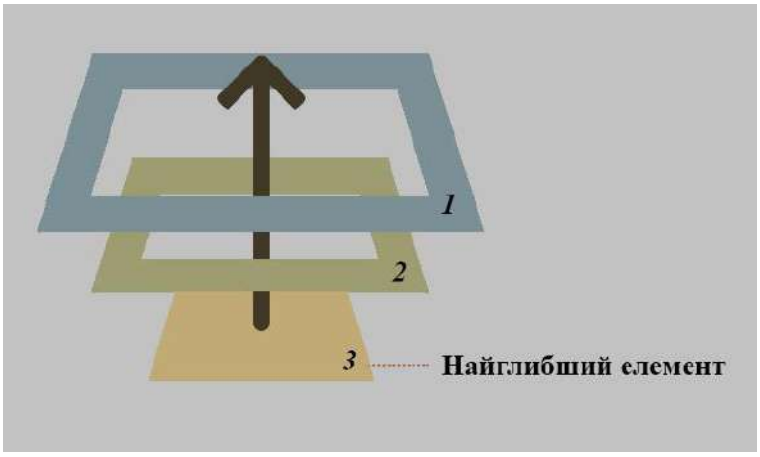


Рисунок 3.6 – Демонстрація спливання подій за ієрархією вкладеності

Оскільки в прикладі вище клацнути на P, то послідовно виведуться alert: p → div → form.

Цей процес називається випливанням, тому що події «спливають» від внутрішнього елемента вгору через батьків, подібно до того, як спливає бульбашка повітря у воді.

На якому б елементі ми не зловили подію, завжди можна дізнатися, де саме вона відбулася. Найглибший елемент, який викликає подію, називається «цільовим» або «вихідним» елементом і доступний як event.target.

Відмінності від this (= event.currentTarget):

event.target – це вихідний елемент, на якому відбулася подія, в процесі спливання він незмінний;

this – це поточний елемент, до якого дійшло спливання, на ньому зараз виконується оброблювач.

Наприклад, якщо стоїть тільки один обробник form.onclick, то він «зловить» всі кліки всередині форми. Де б не був клік всередині – він спливе до елемента <form>, на якому спрацює обробник.

This (= event.currentTarget) завжди буде сама форма, оскільки обробник спрацював на ній. event.target міститиме посилання на конкретний елемент всередині форми, самий вкладений, на якому стався клік.

Можлива і ситуація, коли event.target і this – один і той самий елемент, наприклад, якщо у формі немає інших тегів і клік був на самому елементі <form>.

Спливання йде прямо наверх. Зазвичай подія спливатиме наверх і наверх, до елемента <html>, а потім до document, а іноді навіть до window, викликаючи всі обробники на своєму шляху.

Але будь-який проміжний обробник може вирішити, що подія повністю оброблена, і зупинити спливання.

Для зупинки спливання потрібно викликати метод event.stopPropagation ().

Наприклад, тут із натисканням на кнопку обробник body.onclick не спрацює:

<body onclick = "alert ('сюди обробка не дійде')">
<button onclick = "event.stopPropagation ()"> Натисни мене </ button>
</ body>
event.stopImmediatePropagation ().

Якщо в елемента є кілька обробників на одну подію, то навіть з припиненням спливання всі вони будуть виконані.

Тобто, stopPropagation перешкоджає просуванню події далі, але на поточному елементі всі обробники відпрацюють.

Для того, щоб повністю зупинити обробку, сучасні браузери підтримують метод event.stopImmediatePropagation (). Він не тільки запобігає спливанню, а й зупиняє обробку подій на поточному елементі.

Спливання – це зручно. Не припиняйте його без явної потреби, очевидної і архітектурно прозорої. Найчастіше припинення спливання створює свої підводні камені, які потім доводиться обходити.

Наприклад:

1. Ми робимо меню. Воно обробляє кліки на своїх елементах і робить для них stopPropagation. Начебто, все працює.

2. Пізніше ми вирішили відстежувати всі кліки у вікні, для якоїсь своєї функціональності, наприклад, для статистики, – де взагалі у нас натискають люди. Наприклад, Яндекс. Метрика так робить, якщо підключити відповідну опцію.

3. Над областю, де кліки вбиваються stopPropagation, статистика працювати не буде! Вийшла «мертва зона».

Проблема в тому, що stopPropagation вбиває будь-яку можливість відстежити подія зверху. У сучасному стандарті, крім «спливання» подій, передбачено ще й «занурення». Воно набагато менш затребуване, але іноді, дуже рідко, знання про нього може бути корисним.

Строго кажучи, стандарт виділяє цілих три стадії проходу події:

1. Подія спочатку йде зверху вниз. Ця стадія називається «стадія перехоплення» (capturing stage).

2. Подія досягла цільового елемента. Це – «стадія мети» (target stage).

3. Після цього подія починає спливати. Це – «стадія спливання» (bubbling stage).

У стандарті DOM Events 3 це продемонстровано так:

– занурення (рис. 3.7).

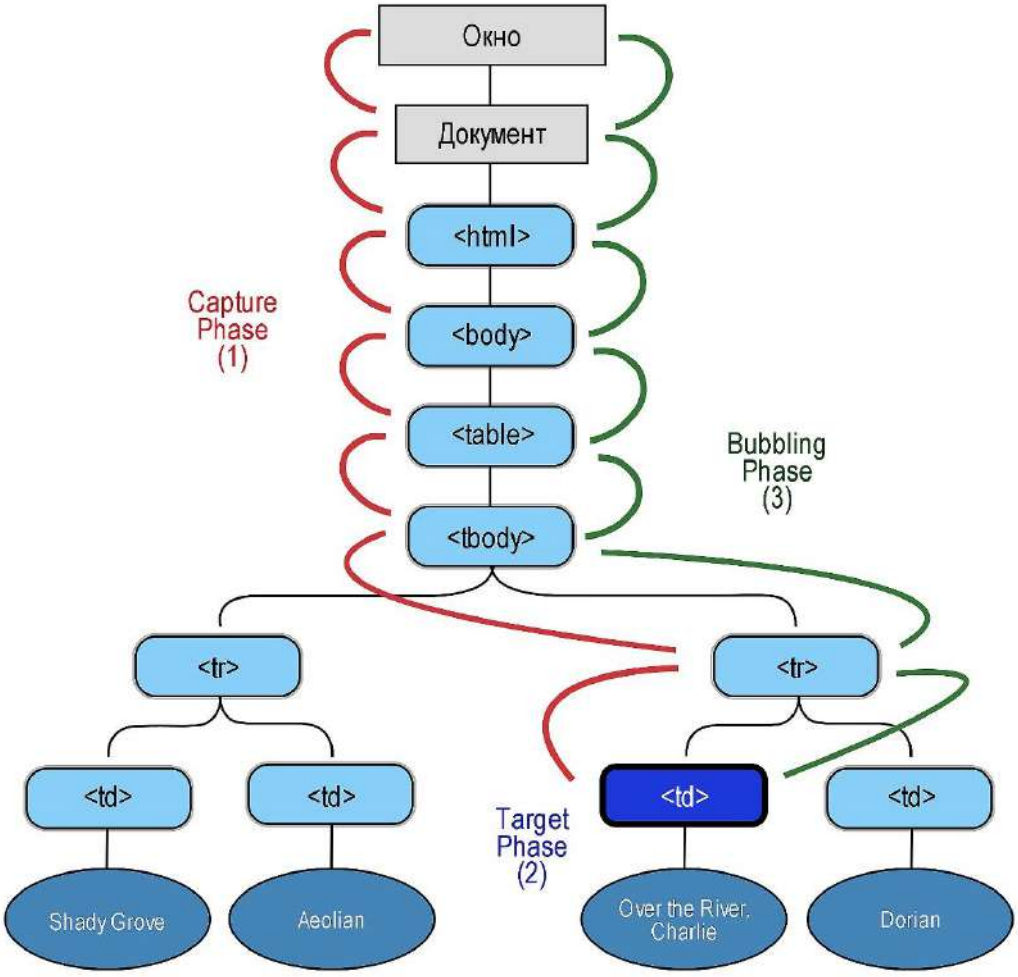


Рисунок 3.7 – Демонстрація занурення і спливання

Тобто, з атисканням на TD подія подорожує по ланцюжку батьків спочатку вниз до елемента («занурюється»), а потім нагору («спливає»), по шляху задіюючи обробники.

Раніше ми говорили тільки про спливання, тому що інші стадії, як правило, не використовуються і проходять непомітно для нас.

Обробники, додані через on...-властивість, нічого не знають про стадії перехоплення, а починають працювати зі спливання.

Щоб зловити подію на стадії перехоплення, потрібно використовувати третій аргумент методу addEventListener.

Якщо аргумент true, то подія буде перехоплена по дорозі вниз.

Якщо аргумент false, то подія буде спіймана із спливанням.

Стадія мети, позначена на рис. 3.7 цифрою (2), особливо не обробляється, оскільки обробники, які призначаються обома цими способами, спрацьовують також на цільовому елементі. Є події, які не спливають, але які можна перехопити.

Бувають події, які можна зловити тільки на стадії перехоплення, а на стадії спливання – ні.

Наприклад, така подія фокусування на елементі onfocus. Звичайно, це велика рідкість, таке виключення існує з історичних причин.

У прикладі нижче на <form>, , <p> стоять ті самі обробники, що і раніше, але на цей раз – на стадії занурення. Щоб побачити перехоплення в дії, клікніть у ньому на елементі <p>.

Обробники спрацюють в порядку «зверху-вниз»: FORM → DIV → P.

Ніхто не заважає призначити обробники для обох стадій, ось так:

натискаючи по внутрішньому елементу <p>, щоб побачити порядок проходу події.

Має бути FORM → DIV → P → P → DIV → FORM. Зауважимо, що елемент <p> бере участь в обох стадіях.

Один і той самий обробник можна призначити на різні стадії. При цьому номер поточної стадії він, за необхідності, може отримати з властивості event.eventPhase (= 1, якщо занурення, = 3, якщо спливання).

Приклади:

```
var elems = document.querySelectorAll ( 'form, div, p');
// на кожен елемент повісити обробник на стадії перехоплення
for (var i = 0; i <elems.length; i ++) {
 elems [i] .addEventListener ( "click", highlightThis, true);
}
var elems = document.querySelectorAll ( 'form, div, p');
for (var i = 0; i <elems.length; i ++) {
 elems [i] .addEventListener ( "click", highlightThis, true);
 elems [i] .addEventListener ( "click", highlightThis, false);
}
```

Звернемо увагу, що з призначенням обробника через on-властивість у нас є this, тому event.currentTarget, як правило, не потрібно, а ось з призначенням через attachEvent обробник не отримує this, так що поточний елемент, якщо потрібно, можна буде взяти лише з замикання .

Для зупинки спливання використовується код event.cancelBubble = true.

Далі ми будемо використовувати стандартні властивості і виклики, оскільки додавання цих рядків, що забезпечують сумісність – досить проста і очевидна задача. Хотілося б зауважити – ці відмінності потрібно знати під час написання JS-коду з підтримкою IE8- без фреймворків. Майже всі JS-фреймворки забезпечують кросбраузерну підтримку target, currentTarget і stopPropagation ().

Алгоритм:

– з настанням події – елемент, на якому вона відбулася, позначається як «цільової» (event.target).

Далі подія спочатку рухається вниз від кореня документа до event.target, по шляху викликаючи обробники, встановлення через

addEventListener(...., true).

Далі подія рухається від event.target вгору до кореня документа, по шляху викликаючи обробники, встановлені через on* і

addEventListene(...., false).

Кожен обробник має доступ до властивостей події:

```
elem.onclick = function (event) {
event = event || window.event;
var target = event.target || event.srcElement;
 // ... тепер у нас є об'єкт події і target
 ...
}
event.stopPropagation? event.stopPropagation(): (event.cancelBubble=true);
(Використовується тернарний оператор).
```

Ще раз хотілося б зауважити: ці відмінності потрібно знати під час написання JS-коду з підтримкою IE8- без фреймворків. Майже всі JS-фреймворки забезпечують кросбраузерну підтримку target, currentTarget і stopPropagation ().

З настанням події елемент, на якому вона відбулася, позначається як «цільовий» (event.target).

Далі подія спочатку рухається вниз від кореня документа до event.target, по шляху викликаючи обробники, встановлені через addEventListener (...., true).

Далі подія рухається від event.target вгору до кореня документа, по шляху викликаючи обробники, встановлені через on * і addEventListener (...., false).

Кожен обробник має доступ до властивостей події:

```
elem.onclick = function (event) {
event = event || window.event;
var target = event.target || event.srcElement;
 // ... тепер у нас є об'єкт події і target
 ...
}
event.stopPropagation? event.stopPropagation():(event.cancelBubble= true);
// використовується тернарний оператор (якщо ... то ....).
```

Будь-який обробник може зупинити подію викликом event.stopPropagation (), але робити це не рекомендується, оскільки в подальшому ця подія може знадобитися, іноді для найнесподіваніших речей.

У сучасній розробці стадія занурення використовується дуже рідко.

Цьому є дві причини:

– історична – оскільки IE лише з версії 9 повною мірою підтримує сучасний стандарт;

– розумна, коли відбувається подія, то розумно дати можливість першому спрацювати оброблювачу на самому елементі, оскільки він найбільш конкретний. Код, який поставив обробник саме на цей елемент, знає максимум деталей про те, що це за елемент, чим він займається.

Далі має сенс передати обробку події батькові – він теж розуміє, що відбувається, але вже менш детально, далі – вище, і так далі, до самого об’єкта document, обробник на якому реалізовує саму загальну функціональність рівня документа.

3.4. Поняття «делегування подій»

Спливання подій дозволяє реалізувати один з найважливіших прийомів розробки – делегування.

Він полягає в тому, що якщо у нас є багато елементів, події на яких потрібно обробляти схожим чином, то замість того, щоб призначати обробник кожному, ми ставимо один обробник на їх загального предка. З нього можна отримати цільовий елемент event.target, зрозуміти, на якому саме нащадку відбулася подія і обробити його.

Розглянемо схематично такий код.

```
<Table>
 <Tr>
 <Th colspan = "3"> <em> Bagua </ em> Chart: Direction, Element, Color, Meaning </ th>
 </ Tr>
 <Tr>
 <Td> ... <strong> Northwest </ strong> ... </ td>
 <Td> ... </ td>
 <Td> ... </ td>
 </ Tr>
 <Tr> ... ще 2 рядки такого ж виду ... </ tr>
 <Tr> ... ще 2 рядки такого ж виду ... </ tr>
</ Table>
```

Наше завдання – реалізувати підсвічування осередку <td> з натисканням.

Замість того, щоб призначати обробник для кожного осередку, яких може бути дуже багато, ми повісимо єдиний обробник на елемент <table>. Він використовуватиме event.target, щоб отримати елемент, на якому відбулася подія, і підсвітити його. Такому коду немає різниці, скільки клітинок у таблиці. Оброблювач все одно один. Ми можемо додавати, видаляти <td> з таблиці, змінювати їхню кількість – підсвічування буде стабільно працювати, оскільки обробник стоїть на <table>.

Втім, у поточній версії коду є недолік. Клік може бути не на тому тегу, який нас цікавить, а всередині нього.

У нашому випадку, якщо поглянути на HTML таблиці уважно, видно, що осередок містить вкладені теги, наприклад .

Природно, клік може відбутися всередині <td>, на елементі . Такий клік буде спійманий єдиним оброблювачем, але target у нього буде не <td>, а .

Усередині обробника table.onclick ми повинні по event.target розібратися, в якому саме <td> був клік.

Для цього ми, використовуючи посилання parentNode, будемо йти вгору по ієрархії батьків від event.target і вище і перевіряти:

– якщо знайшли <td>, це те, що потрібно;

```
var selectedTd;
table.onclick = function (event) {
var target = event.target; // де був клік?
if (target.tagName! = 'TD') return; // нема на TD? тоді не цікавить
highlight (target); // підсвітити TD
```

```
};
function highlight (node) {
 if (selectedTd) {
 selectedTd.classList.remove ( 'highlight');
 }
 selectedTd = node;
 selectedTd.classList.add ( 'highlight');
}.
```

Природно, клік може відбутися всередині <td>, на елементі . Такий клік буде спійманий єдиним оброблювачем, але target у нього буде не <td>, а

```
<Td>
 <Strong> Northwest </ strong>
 ... Metal..Silver..Elders ...
</ Td>.
```

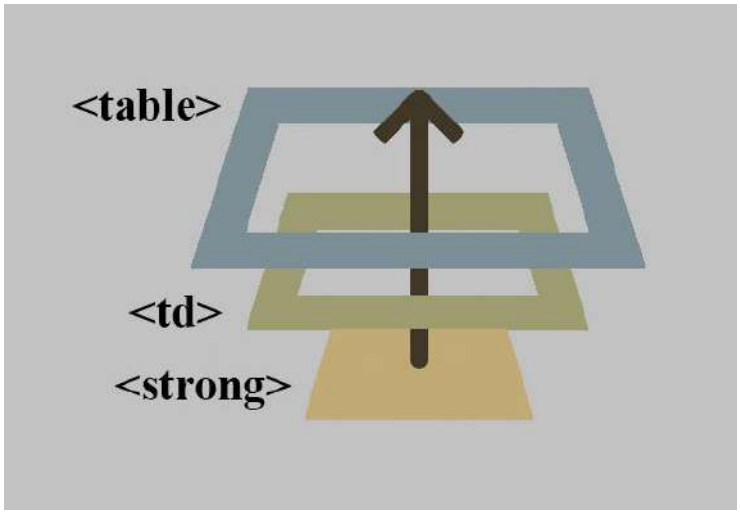


Рисунок 3.8 – Демонстрація занурення і спливання

До речі, в перевірці while можна було б використовувати this замість table.

Це теж працюватиме, оскільки в обробнику table.onclick значенням this є поточний елемент, тобто table.

Можна для цього використовувати і метод closest, за підтримки браузером:

– зазвичай делегування – це засіб оптимізації інтерфейсу. Ми використовуємо один

```
table.onclick = function (event) {
 var target = event.target;
 // цикл рухається вгору від target до батьків до table
 while (target! = table) {
```

```
 if (target.tagName == 'TD') {
 // знайшли елемент, який нас цікавить!
 highlight (target);
 return;
 }
 target = target.parentNode;
 }
 // можлива ситуація, коли клік був поза <td>
 // якщо цикл дійшов до table і нічого не знайшов,
 // то обробник просто закінчує роботу
}
while (target! = this) {
 // ...
}
table.onclick = function (event) {
 var target = event.target;
 var td = target.closest ( 'td');
if (! td) return; // клік поза <td>, не цікавить
 // якщо клік на td, але поза цією таблиці (можливо при вкладених таблицях)
 // то не цікавить
 if (! table.contains (td)) return;
 // знайшли елемент, який нас цікавить!
 highlight (td);
}.
```

Зазвичай делегування – це засіб оптимізації інтерфейсу. Ми використовуємо один table.onclick = function (event) {

```
 var target = event.target;
 // цикл рухається вгору від target до батьків до table
 while (target! = table) {
 if (target.tagName == 'TD') {
 // знайшли елемент, який нас цікавить!
 highlight (target);
 return;
 }
 target = target.parentNode;
 }
 // можлива ситуація, коли клік був поза <td>
 // якщо цикл дійшов до table і нічого не знайшов,
 // то обробник просто закінчує роботу
}
while (target! = this) {
 // ...
```

```
}
table.onclick = function (event) {
 var target = event.target;
 var td = target.closest ( 'td');
if (! td) return; // клік поза <td>, не цікавить
 // якщо клік на td, але поза цією таблицею (можливо при вкладених таблицях)
 // то не цікавить
 if (! table.contains (td)) return;
 // знайшли елемент, який нас цікавить!
 highlight (td);
}.
```

Розглянемо обробник для подібних дій на однотипних елементах.

Вище ми це робили для обробки кліків на <td>.

Але делегування дозволяє використовувати обробник і для абсолютно різних дій. Наприклад, нам потрібно зробити меню з різними кнопками: «Зберегти», «Завантажити», «Пошук» і т.д.

І є об’єкт з відповідними методами: save, load, search і т.п ...

Перше, що може прийти в голову – це знайти кожну кнопку і призначити їй свій обробник серед методів об'єкта. Але більш витончено вирішити задачу можна шляхом додавання одного обробника на все меню, а для кожної кнопки в спеціальному атрибуті, який ми назвемо data-action (можна придумати будь-яку назву, але data-* Є дійсним в HTML5), вкажемо, що вона має викликати:

Оброблювач зчитує вміст атрибута і виконує метод. Погляньте на робочий приклад:

```
 <buton data-action = "save"> Натисніть, щоб Зберегти </ button>
<div id = "menu">
 <button data-action = "save"> Зберегти </ button>
 <button data-action = "load"> Завантажити </ button>
 <button data-action = "search"> Пошук </ button>
</ div>
<script>
 function Menu (elem) {
 this.save = function () {
 alert ( 'зберігаю');
 };
 this.load = function () {
 alert ( 'завантажую');
 };
 this.search = function () {
```

```
alert ( 'шукаю');
};
var self = this;
elem.onclick = function (e) {
var target = e.target;
var action = target.getAttribute ( 'data-action ');
if (action) {
self [action] ();
}
};
}
new Menu (menu);
</ script>.
```

Зверніть увагу, як використовується трюк з var self = this, щоб зберегти посилання на об’єкт Menu. Інакше обробник просто б не зміг викликати методи Menu, тому що його власний this.

Що в цьому випадку нам дає використання делегування подій?

Мабуть, це один з найкорисніших прийомів для роботи з DOM. Він відмінно підходить, якщо є багато елементів, обробка яких дуже схожа.

Алгоритм:

1. Вішаємо обробник на контейнер.

2. В обробнику: отримуємо event.target.

3. В обробнику: якщо event.target або один з його батьків в контейнері (this) – інтереси нас елемент – обробити його.

Звичайно, у делегування подій є свої обмеження.

Не потрібно писати код, щоб привласнити обробник кожній кнопці. Менше коду, менше часу, витраченого на ініціалізацію.

Структура HTML стає по-справжньому гнучкою. Ми можемо додавати / видаляти кнопки в будь-який час.

Даний підхід є семантичним. Також можна використовувати класи .action-save, .action-load замість атрибута data-action.

3.5. Введення в jQuery

jQuery – бібліотека JavaScript, що містить готові функції мови JavaScript, всі операції jQuery виконуються з коду JavaScript [5–9].

Бібліотека jQuery проводить маніпуляції з html-елементами, керуючи їхніми.

Спочатку проводиться обгортання примірника document в функцію jQuery (), після застосовується метод ready (), якому передається функція function () {...}, що

виконується після завантаження документа. На практиці зазвичай використовується скорочена форма такого запису jQuery (function () {...}) ;, або $ (function () {...}) ;. Для зберігання інформації під час роботи з бібліотекою jQuery використовуються змінні JavaScript. У змінних можуть зберігатися елементи. Імена змінних, призначених для зберігання елементів, що повертаються, починаються зі знака $, наприклад: $ h = $ (". List"). Parent (). Parent (). Detach (); Для зберігання декількох елементів використовуються масиви JavaScript: $ k [3] = 15;

Далі наведені правила роботи з бібліотекою jQuery

Додати бібліотеку jQuery на свою веб-сторінку можна двома способами: Використовувати версію файлу jQuery, розміщену на ресурсах Google, Microsoft або jQuery.com. Даний метод використовує «Мережа дистрибуції контенту» (CDN, content distribution network), тобто файл jQuery розташований на іншому веб-сайті, який із запитом користувачем відправляє даний файл на його комп'ютер. Очевидні переваги даного способу – зниження навантаження на власний веб-сервер і прискорення завантаження файлу через розгалуженість мережі серверів дистриб'ютора. Щоб скористатися таким способом, необхідно перейти по одному з посилань: Google CDN, Microsoft CDN, CDNJS CDN.

Після переходу на сайт ресурсу вам буде потрібно лише скопіювати посилання на jQuery-файл і додати його на свою веб-сторінку між тегами <script> ... </ script>. В результаті у вас має вийти, наприклад, <script src = "// ajax.googleapis.com/ajax/libs/jquery/1.11.0/jquery.min.js"> </ script>

Завантажити останню версію бібліотеки jQuery з офіційного сайту. Для цього вам необхідно перейти за адресою jQuery.comі вибрати цікаву для вас версію бібліотеки. Для завантаження пропонується дві версії jQuery-файлу – мінімізований і нестислий. Розмір нестислого файлу близько 300 Кб, він містить коментарі, тому його краще використовувати з метою розробки та налагодження кода. Мінімізована версія файлу важить близько 100 Кб, в ній видалені всі коментарі і непотрібні пропуски, що прискорює завантаження файлу браузером. Для завантаження потрібно перейти за посиланням, і у вікні, натиснути правою кнопкою мишки і вибрати «Зберегти як ...». Після цього помістити файл у потрібну папку (зазвичай для цього використовується папка «scripts») і додати його на вашу сторінку: <script src = "scripts / jquery-2.0.0.min.js"> </ script>

Правила додавання jQuery на сторінку

Розміщуйте посилання на jQuery-файл всередині тега <head>. Розміщуйте посилання на jQuery-файл після посилань на стилі CSS, адже часто бібліотека jQuery проводить маніпуляції зі стилями елементів веб-сторінки.

Розміщуйте інші теги <script> ... </ script> тільки після посилання на файл jQuery, якщо даний скрипт використовує бібліотеку jQuery.

Створення нового html-елемента

Створити порожній html-елемент, наприклад, блок, можна кількома способами:

1) за допомогою короткої форми запису $ (" ")

2) за допомогою інструкції $ (" </ div>")

3) за допомогою інструкції $ (" ")

Всі три способи робочі, але, тим не менш, рекомендується включати відкриваючі та закриваючі теги, щоб показати, що даний елемент може містити інші елементи. За створення нового елемента методу $ () можна передати другий параметр у вигляді об'єкта JavaScript, що визначає додаткові атрибути елемента: $ ("<img />", {src: "images / flower.jpg", title: "Rose_in_garden", click: function () {...}}). appendTo ("body"); Отже створюється елемент <img> з заданими атрибутами і включається в дерево DOM.

3.6. Контрольні запитання та завдання

1. Що таке DOM - модель сайту?
2. Що являє собою метод insertBefore, appendChild?
3. Як здійснюється доступ до елементів HTML -розмітку сайту?
5. Як створюється новий текстовий вузол з даним текстом?
6. Що означає код document.createElement (tag)?
7. Що означає var div = document.createElement ('div')?
8. Що являє собою поняття події?
9. Як створюються обробники подій?
10. Що являє собою делегування подій?
11. Що являють собою методи addEventListener і addRemoveListener?
12. Що являє собою делегування подій?
13. Що являє собою «занурення» і «спливання»?
14. Що являє собою явище перехоплення події?
15. У чому призначення методу getElementsByClassName?
16. Що здійснюється методом querySelector CSS?

3.7. Завдання для самостійного розв’язання

1. Використовуючи наведений нижче код, реалізуйте розміщення футбольного поля по центру екрана, змініть колір фону і реалізуйте обертання м’яча з переміщенням.

```
<! DOCTYPE html>
 <html>
   <head>
    <meta charset = "utf-8" />
     <style>
      #field {
       width: 200px;
       height: 150px;
       border: 10px solid black;
       background-color: # 00ff00;
       position: relative;
       overflow: hidden;
       cursor: pointer;
       }

       #ball {
       position: absolute;
       left: 0;
       top: 0;
       width: 40px;
       height: 40px;
       transition: all 1s;
       }
       </ style>
       </ head>

       <body style = "height: 2000px">
       Натисніть на поле для переміщення м'яча.
       <br />

       <div id = "field">
       <img src = "https://ru.js.cx/clipart/ball.svg" id = "ball" />. . . . . .
       . . . . . .
       . . . . . .
       . . . . . .
       . . . . . .
       . . . . . .
       . . . . . .
       . . . . . .
       </ div>
```

```
<script>
field.onclick = function (event) {
// координати поля щодо вікна браузера
let fieldCoords = this.getBoundingClientRect ();

// м'яч має абсолютне позиціонування (position: absolute), поле - відносне (position: relative)
// таким чином, координати м'яча розраховуються щодо внутрішнього, верхнього лівого кута поля
let ballCoords = {
top:
event.clientY -
fieldCoords.top -
field.clientTop -
ball.clientHeight / 2,
left:
event.clientX -
fieldCoords.left -
field.clientLeft -
ball.clientWidth / 2,
};

// забороняємо перетинати верхню межу поля
if (ballCoords.top <0) ballCoords.top = 0;

// забороняємо перетинати ліву межу поля
if (ballCoords.left <0) ballCoords.left = 0;

// // забороняємо перетинати праву межу поля
if (ballCoords.left + ball.clientWidth> field.clientWidth) {
ballCoords.left = field.clientWidth - ball.clientWidth;
}

// забороняємо перетинати нижню межу поля
if (ballCoords.top + ball.clientHeight> field.clientHeight) {
ballCoords.top = field.clientHeight - ball.clientHeight;
}

ball.style.left = ballCoords.left + 'px';
ball.style.top = ballCoords.top + 'px';
};
</ script>
</ body>
</ html>
```

Рисунок 3.9 – Приклад використання події

2. Внесіть зміни в код, наведений нижче, для додавання трьох прізвищ вставки чотирьох довільних прізвищ після натискання кнопки. Внесіть зміни в стиль оформлення сторінки. Розбийте наведений нижче код на складові стилю і js-код і підключіть їх до html документу окремими файлами.

```
<! DOCTYPE html>
<html>
<head>
<meta charset = "utf-8">
<title> додавання і видалення вузлів </ title>
</ head>
<body>
<ol id = "list">
 <li class = "box1"> <a style="font-size: 36pt"> Іванов І.І. </a> </ li>
 <li class = "box1"> <a style="font-size: 36pt"> Петров С.М. </a> </ li>
 <li class = "box1"> <a style="font-size: 36pt"> Сидоров В.І. </a> </ li>
</ ol>
<script>
function menu ()
{
let newLi = document.createElement ( 'li');
 newLi.innerHTML = 'Вікторов В.В.';
 list.appendChild (newLi);
}
 function menu1 ()
{
 let newLi = document.createElement ( 'li');
 newLi.innerHTML = 'Яковлєва О.Ф.';
 list.insertBefore (newLi, list.children [1])
}
```

```
function removeChildren ()
 {/ * Ваш код * /
// var elem = removeChildren (ol); // очищує список
//elem.innerHTML = '';
if (! Element.prototype.remove) {
Element.prototype.remove = function remove () {
if (this.parentNode) {
this.parentNode.removeChild (this);
}
}
}
var elem = document.body.children [0];
elem.remove ();
}
function removeChildrenNumber () {
var N;
N = prompt ( 'введіть число N', 3)
let liss = list.children [N];
list.removeChild (liss);
}
</ script>
<input class = "box" type = "button" onclick = "menu ()" value = "dobavlenie" />
<input type = "button" onclick = "menu1 ()" value = "вставка" />
<input class = "box2" type = "button" onclick = "removeChildrenNumber ()" value = "Вибіркове видалення" />
<input class = "box3" type = "button" onclick = "removeChildren ()" value = "udalenie" />
</ html>
<style type = "text / css">
.box {
padding: 25px;
 border: 56px solid # d6e9c6;
border-radius: 24px;
color: black;
background-color: blue;
}
.box1 {
padding: 25px;
border: 12px solid # d6e9c6;
border-radius: 14px;
color: green;
background-color: red;
}
.box2 {
padding: 25px;
border: 12px solid # d6e9c6;
```

```
border-radius: 39px;
color: blue;
background-color: yellow;
}
.box3 {
padding: 25px;
border: 12px solid # d6e9c6;
border-radius: 39px;
color: red;
background-color: yellow;
}
.li
{
font-size: 40pt;
border-radius: 18px;
}
.a {
font-size: 36pt;
}
</ style>
```

4. СТВОРЕННЯ ONLINE ВЕБ-АНІМАЦІЇ З ВИКОРИСТАННЯМ CSS

4.1. Правила і властивості CSS-анімації

CSS3-анімація надає сайтам динамічність. Вона оживляє веб-сторінки, покращуючи взаємодію з користувачем. На відміну від CSS3-переходів, створення анімації базується на ключових кадрах, які дозволяють автоматично відтворювати і повторювати ефекти протягом заданого часу, а також зупиняти анімацію всередині циклу.

CSS3-анімація може застосовуватися практично для всіх html-елементів, а також для псевдоелементів :before та :after. Список анімованих властивостей наведено в даному розділі. Зі створенням анімації не варто забувати про можливі проблеми з продуктивністю, оскільки на зміну деяких властивостей потрібно багато ресурсів.

4.1.1. Ключові кадри

Ключові кадри використовуються для вказівки значень властивостей анімації в різних точках анімації. Ключові кадри визначають поведінку одного циклу анімації; анімація може повторюватися нуль або більше разів.

Ключові кадри вказуються за допомогою правила @keyframes, що визначається так:

@keyframes ім'я анімації {список правил}

Створення анімації починається з установки ключових кадрів правила @keyframes. Кадри визначають, які властивості на якому етапі будуть анімовані. Кожен кадр може включати один або більше блоків оголошення з одного або більше пар властивостей і значень. Правило @keyframes містить ім’я анімації елемента, яке пов’язує правило і блок оголошення елемента. Приклад використання правила :

```
@keyframes shadow {
from {text-shadow: 0 0 3px black;}
50% {text-shadow: 0 0 30px black;}
to {text-shadow: 0 0 3px black;}
}.
```

Ключові кадри створюються за допомогою ключових слів from і to (еквівалентні значенням 0% і 100%) Або за допомогою процентних пунктів, яких можна задавати скільки завгодно. Також можна комбінувати ключові слова і процентні пункти. Якщо кадри мають однакові властивості і значення, їх можна об’єднати в одне оголошення:

```
@keyframes shadow {
from {text-shadow: 0 0 3px black;}
50% {text-shadow: 0 0 30px black;}
to {text-shadow: 0 0 3px black;}
},

@keyframes move {
from to {
top: 0;
left: 0;
}
25%,
75% {top: 100%;}
50% {top: 50%;}
}
```

Якщо 0% або 100% кадри не вказані, то браузер користувача створює їх, використовуючи обчислювані (спочатку задані) значення анімімованої властивості.

Якщо кілька правил @keyframes визначені з одним і тим самим ім’ям, спрацює останнім в порядку документа, а всі попередні проігнорує.

Після оголошення правила @keyframes ми можемо посилатися на нього у властивості animation:.

4.1.2. Часова функція для ключових кадрів

Правило стилю ключового кадру також може оголошувати часову функцію, яка має використовуватися під час переміщення анімації до наступного ключового кадру.

```
@keyframes bounce {
from {
top: 100px;
animation-timing-function: ease-out;
}
25% {
top: 50px;
animation-timing-function: ease-in;
```

```
}
50% {
top: 100px;
animation-timing-function: ease-out;
}
75% {
top: 75px;
animation-timing-function: ease-in;
}
to {
top: 100px;
}
}
```

4.1.3. Продовження анімації: властивість animation-duration

Властивість animation-duration визначає тривалість одного циклу анімації. Задається в секундах s або мілісекундах ms. Якщо для елемента задано більше однієї анімації, то можна встановити різний час для кожної, перерахувавши значення через кому.

Властивість не успадковується.

Значення:	**animation-duration**
Час	Вказує час, який анімація займає для завершення одного циклу. Негативні значення недійсні. Якщо час одно 0s, Ключові кадри анімації не діють, але сама анімація відбувається миттєво. Значення за замовчуванням 0s.

Нижче наведено синтаксис анімації

```
animation-duration: .5s;
animation-duration: 200ms;
animation-duration: 2s, 10s;
animation-duration: 15s, 30s, 200ms ;.
```

Часова функція: властивість animation-timing-function

Властивість animation-timing-function описує, як розвиватиметься анімація між кожною парою ключових кадрів. Під час затримки анімації часові функції не застосовуються.

Властивість не успадковується.

Таблиця 4.1 – Значення властивостей

Значення:	**animation-timing-function**
1	2
linear	Лінійна функція, анімація відбувається рівномірно протягом усього часу, без коливань в швидкості .linear
Функції Безьє	
ease	Функція за замовчуванням, анімація починається повільно, розганяється швидко і сповільнюється в кінці. Відповідає cubic-bezier (0.25,0.1,0.25,1).
ease-in	Анімація починається повільно, а потім плавно прискорюється в кінці. Відповідає cubic-bezier (0.42,0,1,1).
ease-out	Анімація починається швидко і плавно сповільнюється в кінці. Відповідає cubic-bezier (0,0,0.58,1).
ease-in-out	Анімація повільно починається і повільно закінчується. Відповідає cubic-bezier (0.42,0,0.58,1).
cubic-bezier (x1, y1, x2, y2)	Дозволяє вручну встановити значення від 0 до 1. На цьому сайті ви зможете побудувати будь-яку траєкторію швидкості зміни анімації.
Покрокові функції	
step-star	Задає покрокову анімацію, розбиваючи анімацію на відрізки, зміни відбуваються на початку кожного кроку. Обчислюється в Steps (1, start).
Step-end	Покрокова анімація, зміни відбуваються в кінці кожного кроку. обчислюється в steps (1, end).
steps (кількість кроків, положення кроку)	Ступінчаста часова функція, яка приймає два параметри. Перший параметр вказує кількість інтервалів у функції. Це має бути позитивне ціле число більше 0, якщо другим параметром не є jump-none – у цьому випадку воно має бути позитивним цілим числом більшим 1. Другий параметр, який є необов'язковим, вказує позицію кроку – момент, в якому починається анімація, використовуючи одне з таких значень:

Установка 3D-перспективи perspective

У нормальному потоці елементи відображаються плоскими і в тій самій площині, що і блок, який містить їх. Двовимірні функції перетворення можуть змінювати зовнішній вигляд елемента, але цей елемент, як і раніше відображається в тій самій площині, що і містить його блок.

Властивості perspective і perspective-origin можна використовувати для додавання відчуття глибини в сцену, роблячи елементи вище по осі Z (ближче до глядача) і удаваними великими, а ті, які знаходяться далі – меншими.

Масштаб пропорційний d / (d – Z), де d – значення перспективи, є відстанню від площині рисування до передбачуваного положення ока глядача.

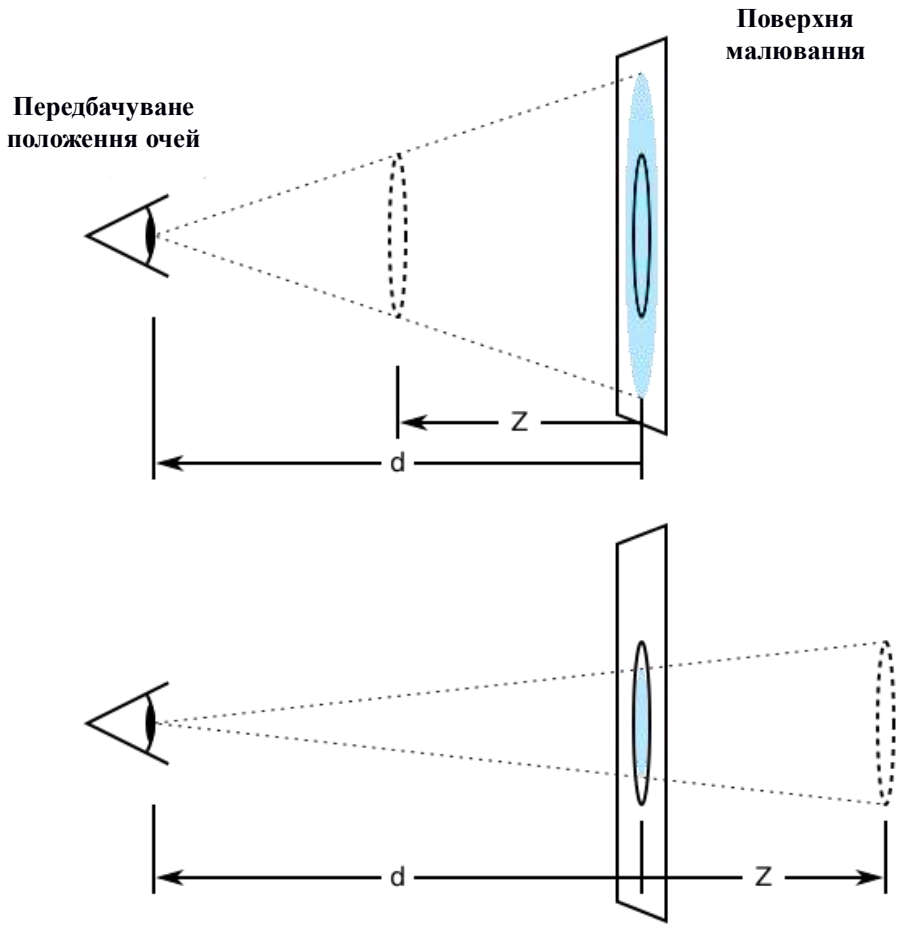


Рисунок 4.1 – Демонстрація перспективи

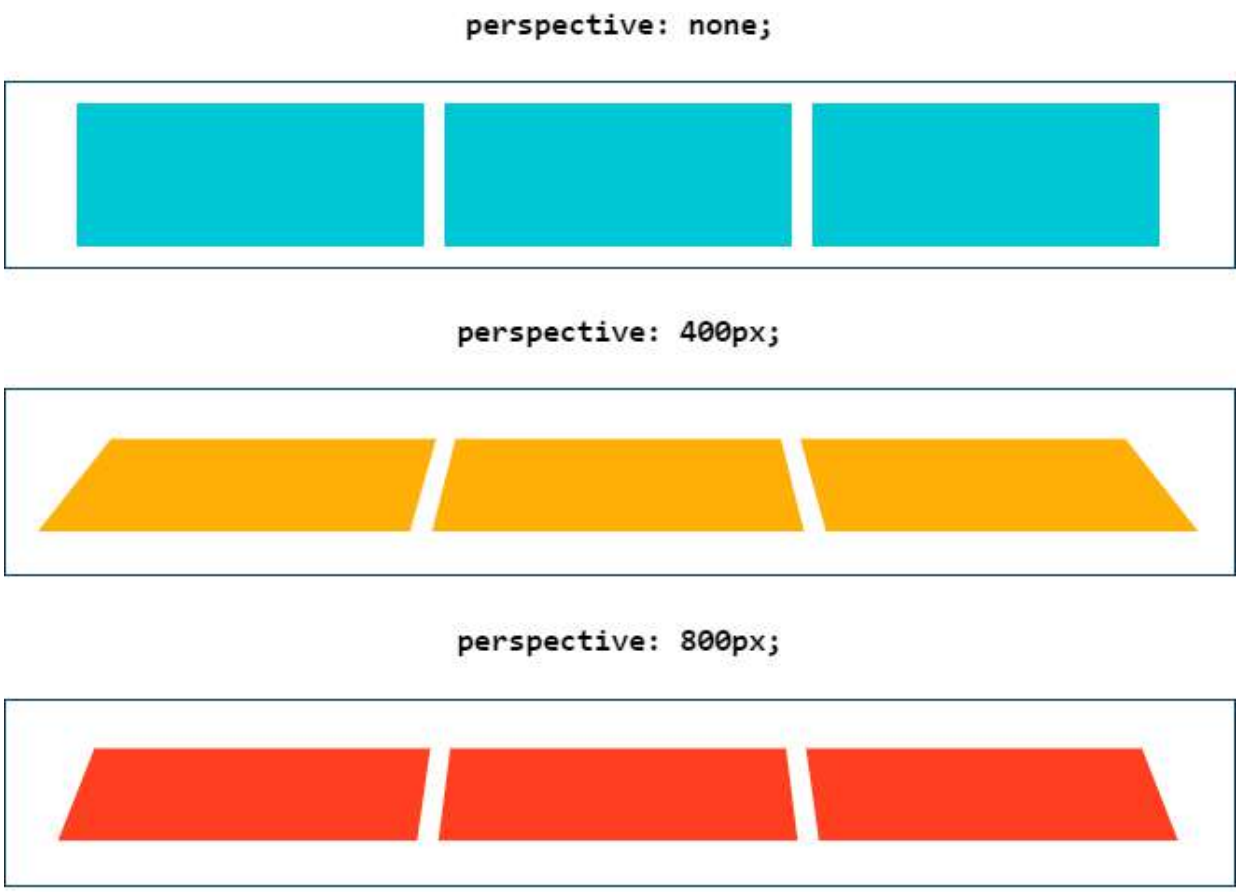
Якщо 3D-перспектива задається за допомогою функції perspective (), 3D-простір активізується тільки для одного елемента. Властивість perspective активує 3D-простір всередині елемента, що містить дочірні трансформовані елементи і застосовується до них. Властивість не успадковується.

Таблиця 4.2 – Значення властивостей

Значення:	Опис переспективи
Довжина	Задає глибину перегляду, тобто відстань по осі Z. Значення може бути будь-яким позитивним числом. Якщо одиниця вимірювання не вказана, за замовчуванням вона вважається в px. Чим більше значення, тим менш виражений ефект. 0 означає відсутність перспективи.
none	Значення за замовчуванням. Означає відсутність перспективи.
initial	Встановлює значення властивості в значення за замовчуванням.
inherit	Успадковує значення властивості від батьківського елемента.

Приклад синтаксису:

```
ul {
 perspective: 500px;
}
li {
 transform: rotateX (50deg);
}
li: hover {
 transform: perspective (900px) rotate3d (180, -45,0, -135deg);
}.
```



Риунок 4.2 – Приклади різних значень 3D-перспективи

Задання точки трансформації для 3D-елемента perspective-origin.

Властивість встановлює точку початку координат для властивості perspective. Значення за замовчуванням perspective-origin: 50% 50% ;. Дозволяє змінювати напрямок трансформації дочірнього 3D-елемента. Властивість має використовуватися разом з властивістю perspective для блоку-контейнера і властивістю transform для дочірнього елемента. Або не успадковується.

Нижче наведено синтаксис стилю:

```
ul {
perspective: 150px;
perspective-origin: 10% 10%;
}
li {
transform: rotateX (50deg);
}
```

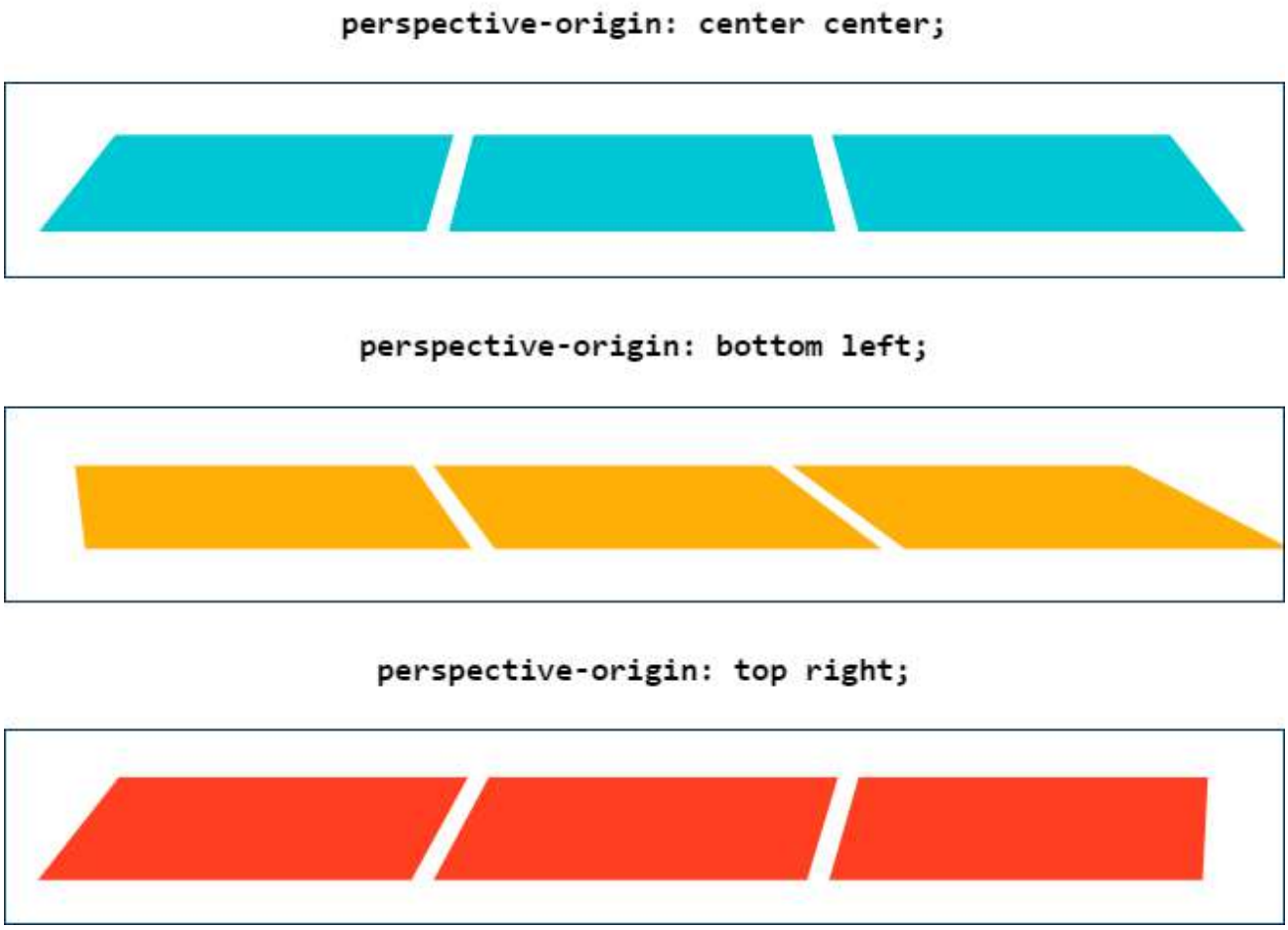


Рисунок 4.3 – Приклади задання точки трансформації

Стиль 3D-перетворень transform-style

Властивість визначає, як дочірні трансформовані елементи (елементи, для яких задано властивість transform) відрисовуваних у тривимірному просторі. Задається для блоку-контейнера. Або не успадковується.

Таблиця 4.3 – Значення властивостей

Значення:	Опис
flat	Значення за замовчуванням. Всі дочірні елементи відображаються плоскими в двовимірній площині блоку-контейнера.
preserve-3d	Має у своєму розпорядженні елементи в тривимірному просторі.
initial	Встановлює значення властивості в значення за замовчуванням.
inherit	Успадковує значення властивості від батьківського елемента.

Приклад коду демонструє властивості перспективи і трансформації

HTML код	Код стилю
<section> *** <kbd>transform-style: flat;</kbd> *** *** *** *** *** *** *** *** <kbd>transform-style: preserve-3d;</kbd> *** *** *** *** *** *** *** ***</section>***	****{margin:0;box-sizing: border-box;}*** ***section {*** ***max-width: 600px;*** ***margin: 0 auto;*** ***text-align: center;*** ***}*** ***p {*** ***padding: 20px 0;*** ***}*** ***.container {*** ***perspective: 500px;*** ***margin: 0 40px;*** ***background: rgba(195,209,212, .7);*** ***border-radius: 5px;*** ***}*** ***.style-flat {*** ***transform-style: flat;*** ***}*** ***.style-3d {*** ***transform-style: preserve-3d;*** ***}*** ***.flip {*** ***height: 160px;*** ***transform: rotateX(45deg);*** ***border-radius: 5px;*** ***background: rgba(169,90,145, .8);*** ***-webkit-animation: rotate 5s infinite;*** ***animation: rotate 5s infinite;*** ***}*** ***@keyframes rotate {*** ***50% {*** ***transform: none;*** ***}*** ***}***

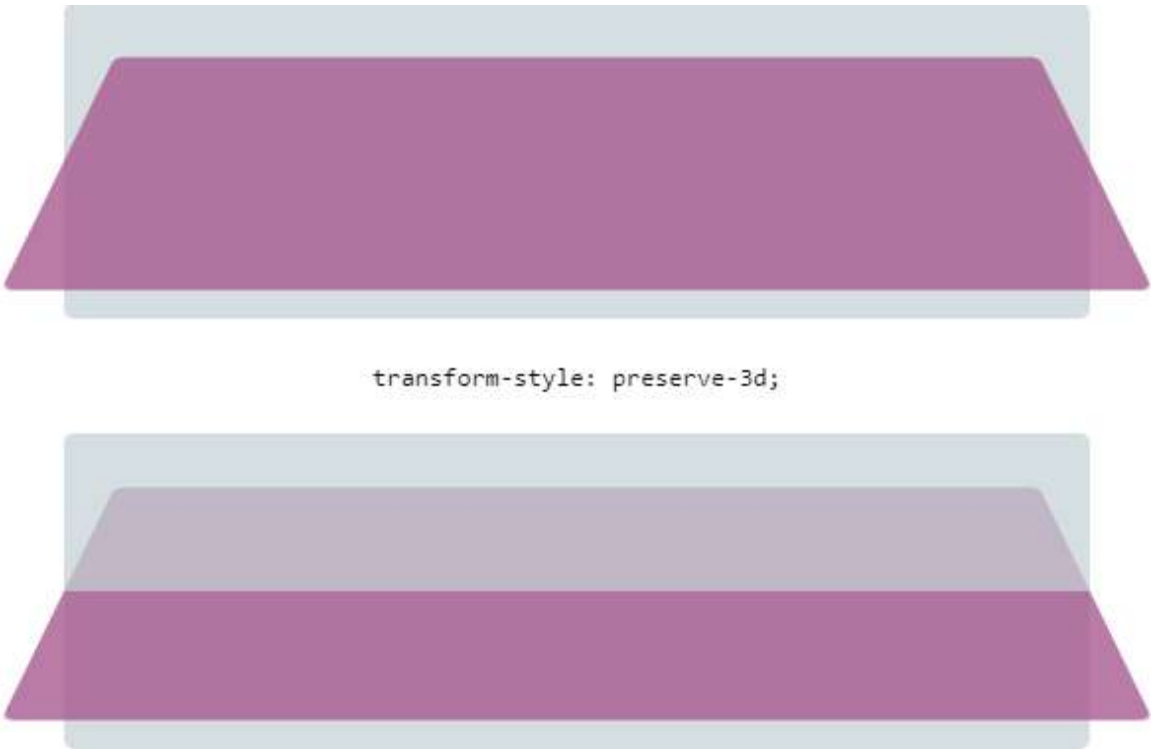


Рисунок 4.4 – Приклад використання властивостей перспективи і трансформації

Таблиця 4.4 – Функції анімації

Функція	**Опис**
1	2
matrix3d (N, n, n, n, n, n, n, n, n, n, n, n, n, n, n, n)	Функція задає тривимірне перетворення як однорідну матрицю розміром 4 × 4 з шістнадцятьма значеннями в стовпчиках. Всі інші функції перетворень засновані на даній функції.
translate3d (x, y, z)	Функція задає переміщення елемента в 3D-просторі. Рух відбувається по вектору [tx, ty, tz], де tx переміщення уздовж осі X, ty - переміщення уздовж осі Y, а tz – уздовж осі Z. Значення можуть задаватися в одиницях довжини або в%. Негативні значення будуть переміщувати елемент в протилежному напрямку. ***transform: translate3d (100px, 100px, -200px);*** ***transform: translate3d (50%, -100%, 10%);*** ***transform: translate3d (-100px, -30px, 50px);***
translateZ (z)	Функція задає переміщення елемента на задану відстань в напрямку осі Z. Значення можуть задаватися в одиницях довжини або в%. Негативні значення будуть переміщувати елемент в протилежному напрямку. ***transform: translateZ (300px);*** ***transform: translateZ (–50%);*** ***transform: translateZ (150%);***

Продовження таблиці 4.4

1	2
scale3d (x, y, z)	Функція задає операцію тривимірного масштабування за векторуом масштабування [sx, sy, sz], описуваним трьома параметрами. Негативні значення відображають елемент дзеркально уздовж трьох осей. ***transform: scale3d (2, 1, 3);*** ***transform: scale3d (-1,-2,-1);***
scaleZ (z)	Функція масштабує елемент в напрямку осі Z, роблячи його більше або менше. Як значення задається число. Результат функції найбільш виражений зі спільним використанням з такими функціями, як rotate () і perspective (). ***transform: scaleZ (3); transform: scaleZ (-1);***
rotate3d (x, y, z, кут)	Функція обертає елемент за годинниковою стрілкою щодо трьох осей. Елемент повертається під кутом, що задається останнім параметром щодо вектора напрямку [x, y, z]. Негативні значення повертають елемент проти годинникової стрілки. ***transform: rotate3d (1, 1, 2, 45deg);***
rotateX (кут)	Функція задає поворот за годинниковою стрілкою під заданим кутом щодо осі X.Функція rotateX (180deg) еквівалентна rotate3d (1,0,0,180deg). ***transform: rotateX (30deg); transform: rotateX (-135deg);***
rotateY (кут)	Функція задає поворот за годинниковою стрілкою під заданим кутом щодо осі Y.Функція rotateY (180deg) еквівалентна rotate3d (0,1,0,180deg). ***transform: rotateY (30deg);*** ***transform: rotateY (-135deg);***
rotateZ (кут)	Функція задає поворот за годинниковою стрілкою під заданим кутом щодо осі Z.Функція rotateZ (180deg) еквівалентна rotate3d (0,0,1,180deg). ***transform: rotateZ (30deg);*** ***transform: rotateZ (-135deg);***
perspective (n)	Функція змінює перспективу огляду елемента, створюючи ілюзію глибини. Чим більше значення функції перспективи, тим далі від наглядача розташований елемент. Значення має бути більше нуля. ***transform: perspective (300);*** ***transform: perspective (300px);***
initial	Встановлює значення властивості в значення за замовчуванням.
inherit	Успадковує значення властивості від батьківського елемента.

Приклад використання властивості трансформації

div {
transform: rotateX (150deg);
}.

CSS flexbox (Flexible Box Layout Module) – модуль макета гнучкого контейнера – являє собою спосіб компонування елементів, в основі лежить ідея осі. Flexbox складається з гнучкого контейнера (flex container) і гнучких елементів (flex items). Гнучкі елементи можуть вибудовуватися в рядок або стовпчик, а вільний простір розподіляється між ними різними способами.

Модуль flexbox дозволяє вирішувати такі завдання:

Розташовувати елементи в одному з чотирьох напрямків: зліва направо, справа наліво, зверху вниз або знизу вгору.

Перевизначати порядок відображення елементів.

1. Автоматично визначати розміри елементів таким чином, щоб вони вписувалися в доступний простір.

2. Вирішувати проблему з горизонтальним і вертикальним центруванням.

3. Переносити елементи всередині контейнера, не допускаючи його переповнення.

4. Створювати колонки однакової висоти.

5. Створювати притиснутий до низу сторінки підвал сайту.

Flexbox вирішує специфічні завдання – створення одновимірних макетів, наприклад, навігаційної панелі, оскільки flex-елементи можна розміщувати тільки по одній з осей.

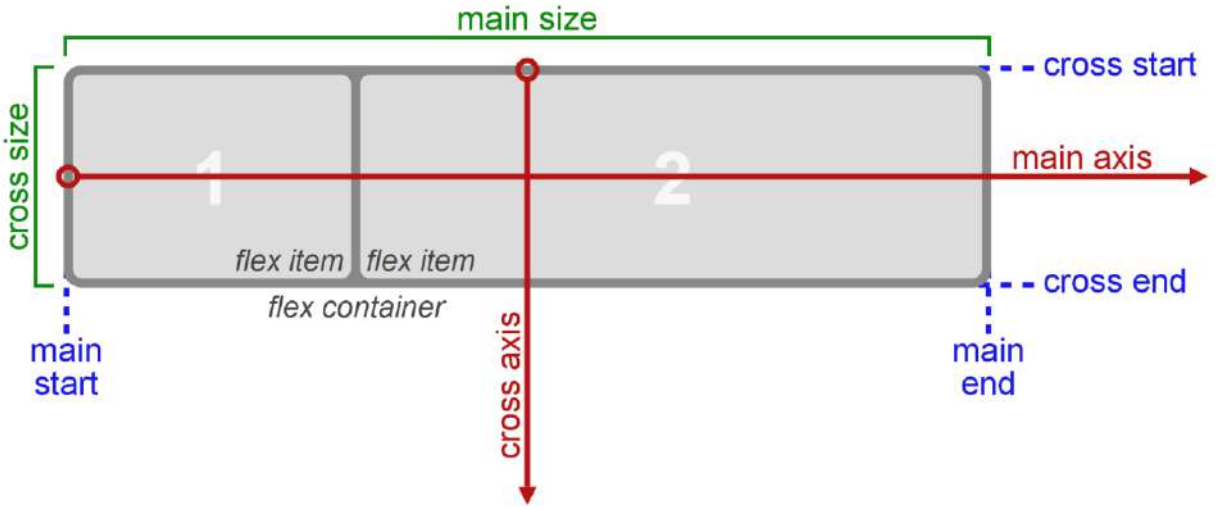


Рисунок 4.5 – Приклад використання властивостей перспективи і трансформації

Для опису модуля Flexbox використовується певний набір термінів. Значення flex-flow і режим запису визначають відповідність цих термінів фізичними напрямками: вгору / вправо / вниз / вліво; осями: вертикальна / горизонтальна; розмірами: ширина / висота.

Головна вісь (main axis) – вісь, уздовж якої викладаються flex-елементи. Вона простягається в основному вимірі.

Main start і main end – лінії, які визначають початковий та кінцевий боки flex-контейнера, відносно яких викладаються flex-елементи (починаючи з main start у напрямку до main end).

Основний розмір (main size) – ширина або висота flex-контейнера чи flex-елементів, залежно від того, що з них знаходиться в основному вимірі, визначають основний розмір flex-контейнера або flex-елемента.

Поперечна вісь (cross axis) – вісь, перпендикулярна головній осі. Вона простягається в поперечному вимірі.

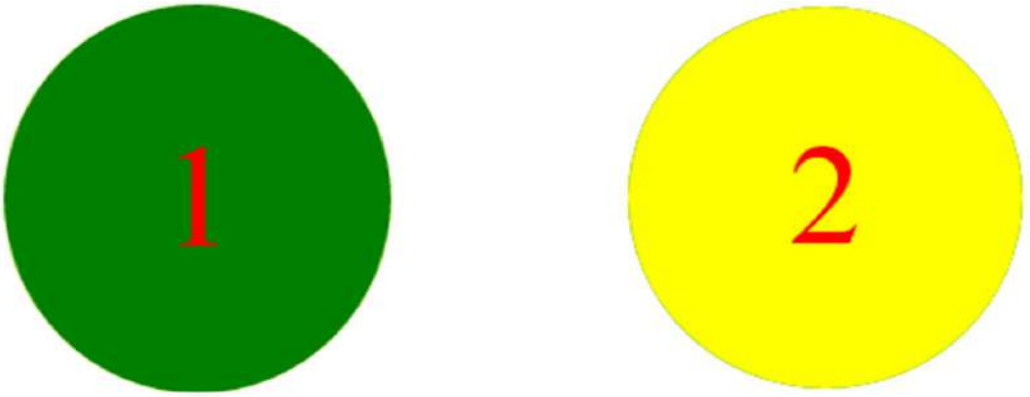
Cross start і cross end – лінії, які визначають початковий та кінцевий боки поперечної осі, відносно яких викладаються flex-елементи.

Поперечний розмір (cross size) – ширина або висота flex-контейнера чи flex-елементів, залежно від того, що знаходиться в поперечному вимірі, є їх поперечним розміром.

4.2. Практичні приклади використання CSS + JS анімації

Розглянемо використання зазначених властивостей і правил на прикладі створення анімаційного сюжету адаптованості для відтворення на мобільних пристроях.

Метою завдання є використання заданого html-коду для створення анімаційного сюжету з обертовим диском зі сторонами показань (рис. 4.6).



Передня (перша) сторона Друга (задня) сторона після повороту

Рисунок 4.6 – Обертовий диск

Опис стилю	Опис розмітки
@keyframes rotate { *from {transform: rotateY (0deg); 0}* *to {transform: rotateY (360deg); }* *}* *@keyframes Zindex {* *from {z-index: 1; }* *to {z-index: 3; }* *}* *body {* *margin: 0;* *}* *#wrapper {* *width: 500px;* *margin: 0 auto;* *}* *.disk {* *position: relative;* *width: 200px;* *height: 200px;* *margin: 0 auto;* *font-size: 75px;* *line-height: 200px;* *text-align: center;* *color: red;* *animation-name: rotate;* *animation-duration: 4s;* *animation-iteration-count: infinite;* *}* *.disk .side {* *position: absolute;* *top: 0;* *left: 0;* *width: 200px;* *height: 200px;* *border-radius: 50%;* *}* *.disk .side.front {* *background: green;* *z-index: 2;* *}* *.disk .side.back {* *transform: rotateY (180deg);* *background: yellow;* *animation-name: Zindex;*	*<! DOCTYPE html>* *<html>* *<head>* *<title> </ title>* *<link rel = "stylesheet" type = "text / css" href = "css / all.css">* *</ head>* *<body>* * * * 1 </ div>* * 2 </ div>* *</ div>* *</ body>* *</ html>*

```
    animation-delay: 2s;
 animation-duration: 4s;
 animation-iteration-count: infinite;
}
```

Довідки на ресурсах www.htmlacademia.ru, www.w3shool.com, http://qaru.site/questions/234673/changing-background-image-with-css3-animations – зміна картинок CSS-3 анімацією, https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animated_properties.

Під час виконання завдання потрібно використовувати наведені нижче рекомендації.

Завантажити в редактор коду sibline код-html. В ході виконання наступного завдання використовувати додатки А, В, С.

Порядок виконання завдання:

– створити файл Laba3.html зі змістом, зазначеним вище з варіантом підключення стилів окремим файлом all.css;

– домогтися відображення в браузері обертання диска навколо вертикального діаметра зі зміною сторін, як зазначено на рис. 4.1;

– домогтися шляхом зміни змісту файла example.html переміщення обертального диска зі зміною зображення;

– змінити стиль зображення так, щоб обертальний диск переміщувався горизонтально і вертикально (на весь розмір екрану) як стрибучий м’яч;

– внести зміни в файл example.html так, щоб з обертанням замість зміни цифр змінювалися графічні зображення, тобто вміст файлу 1.jpg змінювалося на 2.jpg.

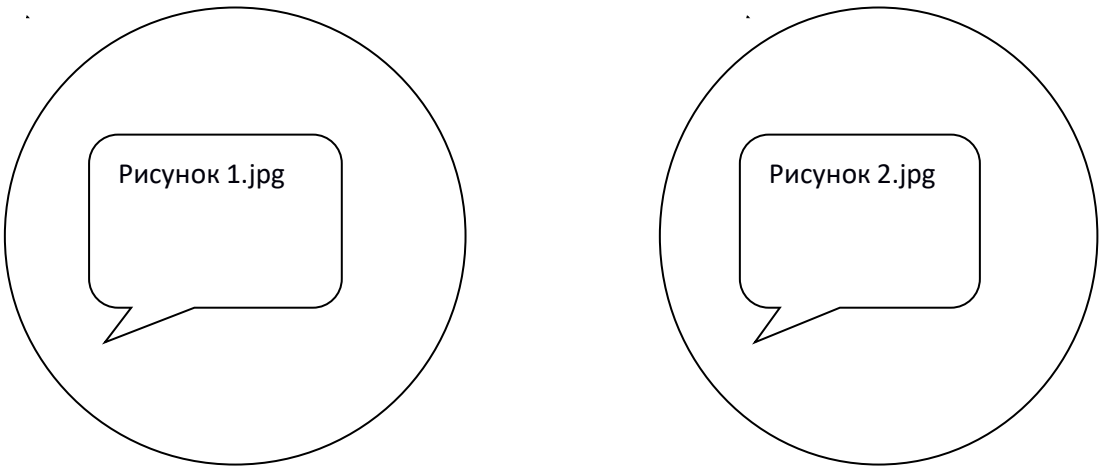


Рисунок 4.7 – Пояснення завдання

Нижче наведений приклад використання CSS-властивості transitions і hover-ефекту. З наведенням мишки на карту вона перевертається. З відведенням мишки вона перевертається назад. З наведенням мишки на першу карту вона піднімається вгору. Друга карта опускається вниз. Зворотний бік карти використовує властивість прозорість.

```
<!doctype html>
<head>
        <meta charset="utf-8">
        <title>css transitions: card flip</title>
        <style type="text/css">

section {
        perspective: 600;
        margin: 10vw;
}
.card,
.front,
.back {
        transition: all 0.4s ease-in-out;
 }
.card {
  width: 200px;
  height: 300px;
  border-radius: 10px;
  margin: auto;
  float: left;
  font-size: .8em;
  position: relative;
}
.card .front {
position: absolute;
top: 0;
left: 0;
z-index: 900;
width: inherit;
height: inherit;
border-radius: 10px;
border: 10px solid #fff;
text-align: center;
```

Успадкування властивостей

```
box-shadow: 0 1px 5px rgba(0,0,0,0.9);
transform: rotatex(0deg) rotatey(0deg);
transform-style: preserve-3d;
backface-visibility: hidden;
background:
linear-gradient(-45deg, white 25%, transparent
25%, transparent 75%, red 75%, red) 0 0,
linear-gradient(-45deg, red 25%, transparent
25%, transparent 75%, white 75%, white) 1em
1em,
linear-gradient(45deg, red 17%, transparent 17%,
transparent 25%, red 25%, red 36%, transparent
36%, transparent 64%, red 64%, red 75%,
transparent 75%, transparent 83%, red 83%) 1em
1em;
background-color: white;
background-size: 2em 2em;
background-position: -1em -1em;

}
.card:hover .front {
  z-index: 900;
  transform: rotatey(180deg);
  box-shadow: 0 15px 50px rgba(0,0,0,0.2);
}
.card .back {
  float: none;
  position: absolute;
  top: 0;
  left: 0;
  z-index: 800;
  width: inherit;
  height: inherit;
  background: #fff;
  text-shadow: 1px 1px 1px rgba(0,0,0,0.6);
  transform: rotatey(-180deg);
  transform-style: preserve-3d;
  backface-visibility: hidden;
  border-radius: 10px;
  border: 10px solid #fff;
    box-shadow: 0 1px 5px rgba(0,0,0,0.9);
```

```
}
.card:nth-of-type(2) {
  z-index: 2000;
}
.card:hover .back {
  z-index: 1000;
  transform: rotatey(0);
}
.card:first-of-type {
  transform: rotate(-5deg);
}
.card:nth-of-type(2):hover{
  transform: rotate(0) translatey(-40px);
}
.card:last-of-type {
  transform: rotate(5deg) translatex(10px);
}
.card:first-of-type:hover{
  transform: rotate(10deg) translatey(30px);
}
.card .back {
        color:red;
        font-size: 50px;
        line-height: 300px;
        text-align:center;
}
.spades .back {
        color: black;
}
.card .back:after  {
        content: 'a';
        position:absolute;
        top: 10px;
        left: 10px;
        line-height:20px;
}
.card .back:before {
        content: 'a';
        position:absolute;
        bottom: 10px;
        right: 10px;
```

```
        line-height:20px;
}
</style>
</head>
<body>
        <section class="cards">
            <div class="card">
                <div class="front"></div>
                <div class="back">♥</div>
            </div>
            <div class="card spades">
                <div class="front"></div>
                <div class="back">♣</div>
            </div>
            <div class="card">
                <div class="front"></div>
                <div class="back">♦</div>
            </div>
        </section>
</body>
</html>
```

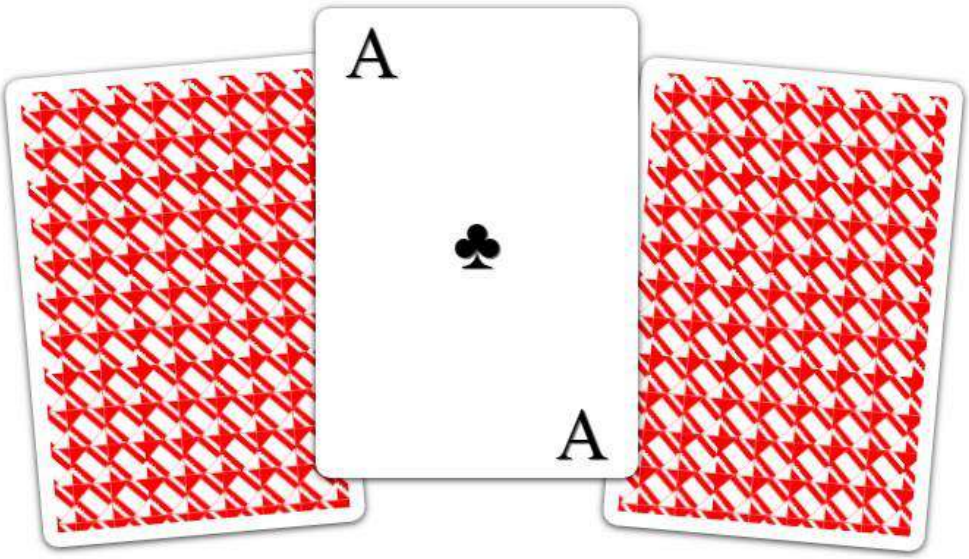


Рисунок 4.8 – Приклад використання властивості CSS і hover ефекту

Приклад використання CSS-властивості transitions і Hover-ефекту. З наведенням мишки на карту вона перевертається. З відведенням мишки вона

перевертається назад. З наведенням мишки на першу карту вона піднімається вгору, друга карта опускається вниз.

Даний приклад демонструє використання двох обробників на одну й ту саму подію. Відкрийте інспектор, щоб побачити журнал подій на вкладці консолі.

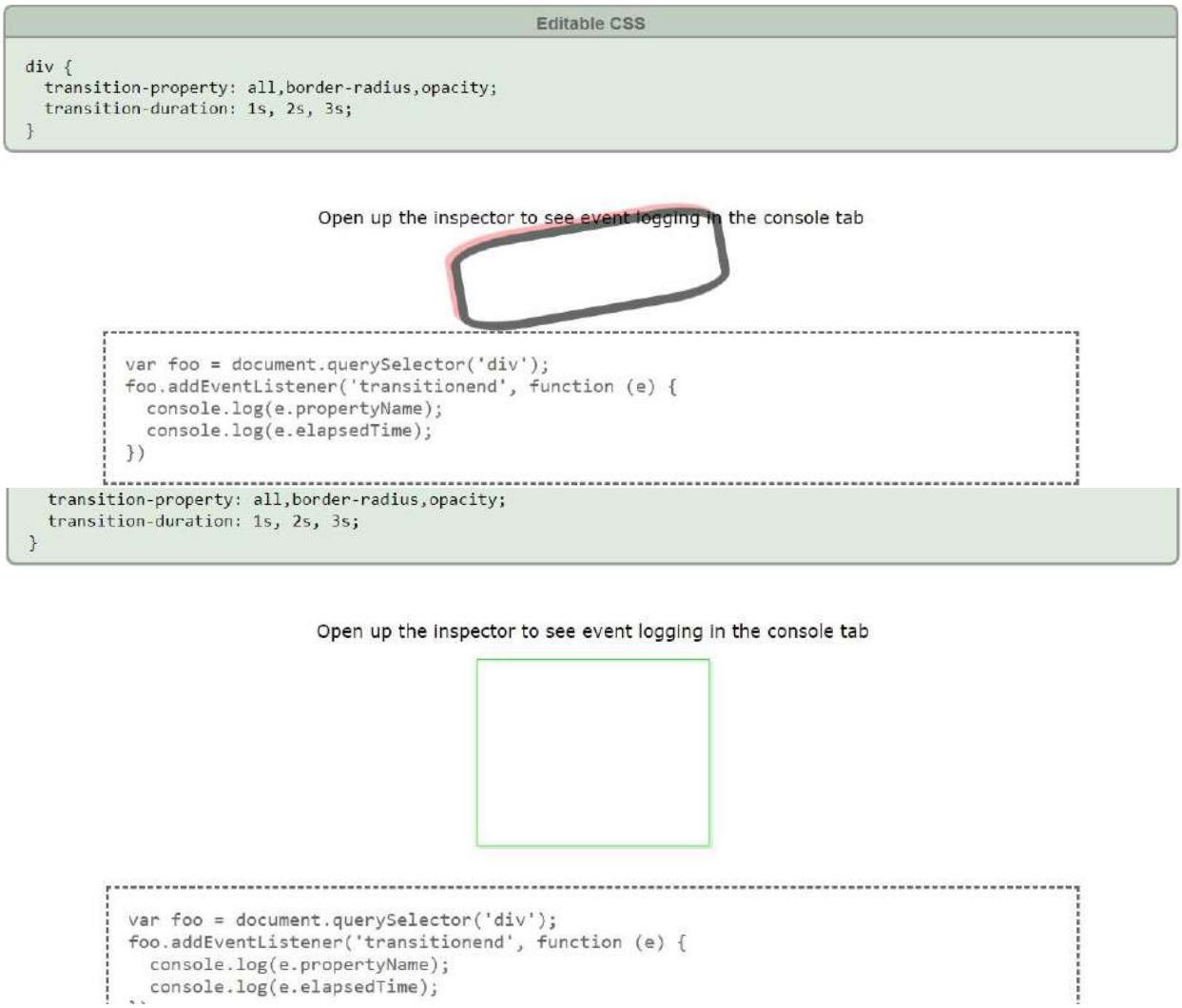


Рисунок 4.9 – Приклад використання перехідних подій
з використанням властивостей CSS + JS і hover ефекту

```
<! doctype html>
<html>
<head>
<meta charset = "utf-8" />
<style type = "text / css">
@import url (../ styles / style-edit.css);
script {
        display: block;
        color: # 666;
        width: 80%;
        margin: 2em auto;
        padding: 1em;
```

```
        border: 3px dashed;
        font: 1.1em consolas, "courier new", courier, monospace;
        white-space: pre-wrap;
}
section {display: flex; flex-direction: column; align-items: center;}
div {
        color: # f00;
        border: 1px solid # 00ff00;
        border-radius: 0;
        transform: scale (1) rotate (0deg);
        opacity: 1;
        box-shadow: 3px 3px rgba (0, 0, 0, 0.1);
        width: 50px;
        padding: 100px;
}
div: hover {
        color: # 000000;
        border: 5px dashed # 000000;
        border-radius: 50%;
        transform: scale (2) rotate (-10deg);
        opacity: 0.1;
        box-shadow: -3px -3px rgba (255, 0, 0, 0.5);
        width: 100px;
        padding: 20px;
}
</ style>
<style contenteditable id = "editme"> div {
 transition-property: all, border-radius, opacity;
 transition-duration: 1s, 2s, 3s;
} </ style>
<title> transitionend events </ title>
</ head>
<body>
<section>
<p> open up the inspector to see event logging in the console tab </ p>
<div> </ div>
</ section>
<script> var foo = document.queryselector ( 'div');
foo.addeventlistener ( 'transitionend', function (e) {
 console.log (e.propertyname);
 console.log (e.elapsedtime);
})
</ script>
</ body>
</ html>
```

Наступний приклад анімує загасання коливань м’яча з киданням вниз.

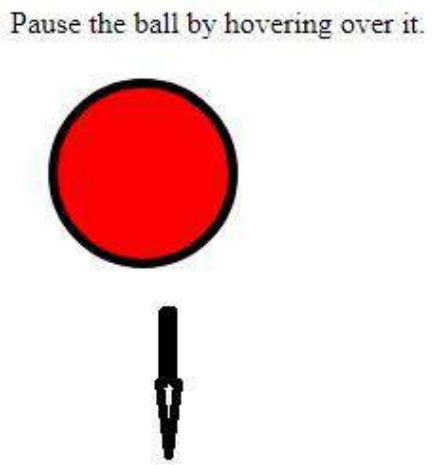


Рисунок 4.10 – Приклад використання перехідних подій
з використанням властивостей CSS і hover- ефекту

```
<title></title>
<meta charset="utf-8">
<link type="text/css" href="../styles/style-edit.css" rel="stylesheet">
<style type="text/css">
div {
        padding: 20px;
        margin: 20px;
        border: 5px black solid!important;
        color: white;
        font-family: sans-serif;
        text-align: center;
        font-size: 2rem;
}
.ball {
        height: 50px;
        width: 50px;
        border-radius: 50%;
        background-color: red;
}
@keyframes bounce {
        0%  {transform: translatey(0);     animation-timing-function: ease-in;}
        30% {transform: translatey(100px); animation-timing-function: ease-in;}
        58% {transform: translatey(200px); animation-timing-function: ease-in;}
        80% {transform: translatey(300px); animation-timing-function: ease-in;}
        95% {transform: translatey(360px); animation-timing-function: ease-in;}
        15%,
        45%,
```

```
        71%,
        89%,
        100% {transform: translatey(380px); animation-timing-function: ease-out;}
}
</style>
<style contenteditable id="editme">.ball {
        animation-name: bounce;
        animation-duration: 5s;
        animation-iteration-count: 1;
        animation-delay: 1s;
        animation-timing-function: ease-in;
}
.ball:hover {
        animation-play-state: paused;
}
</style></head>
<body>
<p>pause the ball by hovering over it.</p>
<div class="ball"></div>
</body>
        </html>
```

Наступний приклад демонструє створення і анімацію 3d об'єкта у вигляді куба, створеного на css і анімованого за допомогою css-коду. У ньому наочно видно опис кожної сторони куба як у розмітки сторінки, так і в його оформленні за допомогою стилів. Особливістю є використання ascii кодів клавіатури для створення анімаційного, хоча його можна створити і автоматично, що буде показано нижче. Є також можливість вписувати зображення на кожну сторону куба.

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
<link type="text/css" rel="stylesheet"
href="csscube.css">
<head>
<body>
<div class="container">
  <div class="cube">
    <div class="side front">front</div>
    <div class="side back">back</div>
    <div class="side right">right</div>
```

```
// обробник події
натискання на
кнопку 37 – код
клавіші – стрілка
вниз
// 38 – код клавіші ->
// 39 – код клавіші
стрілка вгору.
// 40 – код клавіші
стрілка вліво
```

```
    <div class="side left">left</div>
    <div class="side top">top</div>
    <div class="side bottom">bottom</div>
  </div>
                </div>
                <script>
                (function () {
        var rotateY = 0,
                  rotateX = 0;

        document.onkeydown = function (e) {
                  if (e.keyCode === 37) rotateY -= 4
                else if (e.keyCode === 38) rotateX += 4
                else if (e.keyCode === 39) rotateY += 4
                else if (e.keyCode === 40) rotateX -= 4

document.querySelector('.cube').style.transform =
     'rotateY(' + rotateY + 'deg)'+  'rotateX(' + rotateX +
'deg)';
    }
})();
                </script>
                <body>
                        <html>
* {
        box-sizing: border-box;
}

body {
/*
background-image:
url(http://subtlepatterns2015.subtlepatterns.netdna-
cdn.com/patterns/blackorchid.png);
  overflow: hidden;
  */
  background-position: fixed;
   background-color: #c7b39b; /* Цвет фона */
}
.container {
        position: relative;
        margin: 200px auto;
        width: 300px;
        height: 300px;
        perspective: 500px;
}
```

Вміст файлу csscube.css

```
.cube {
	width: inherit;
	height: inherit;
	transform-style: preserve-3d;
}

.side {
	position: absolute;
	width: inherit;
	height: inherit;
	border: 5px solid #fff;
	font: normal 70px Arial;
	text-align: center;
	line-height: 300px;
	color: #fff;
}
.front {
	transform: translateZ(150px);
}
.back {
	transform: rotateY(180deg) translateZ(150px);
}

.right {
	transform: rotateY(90deg) translateZ(150px);
}

.left {
	transform: rotateY(-90deg) translateZ(150px);
}

.top {
	transform: rotateX(90deg) translateZ(150px);
}

.bottom {
	transform: rotateX(-90deg) translateZ(150px);
}
```

Результат виконання даного коду наведено на рис. 4.11.

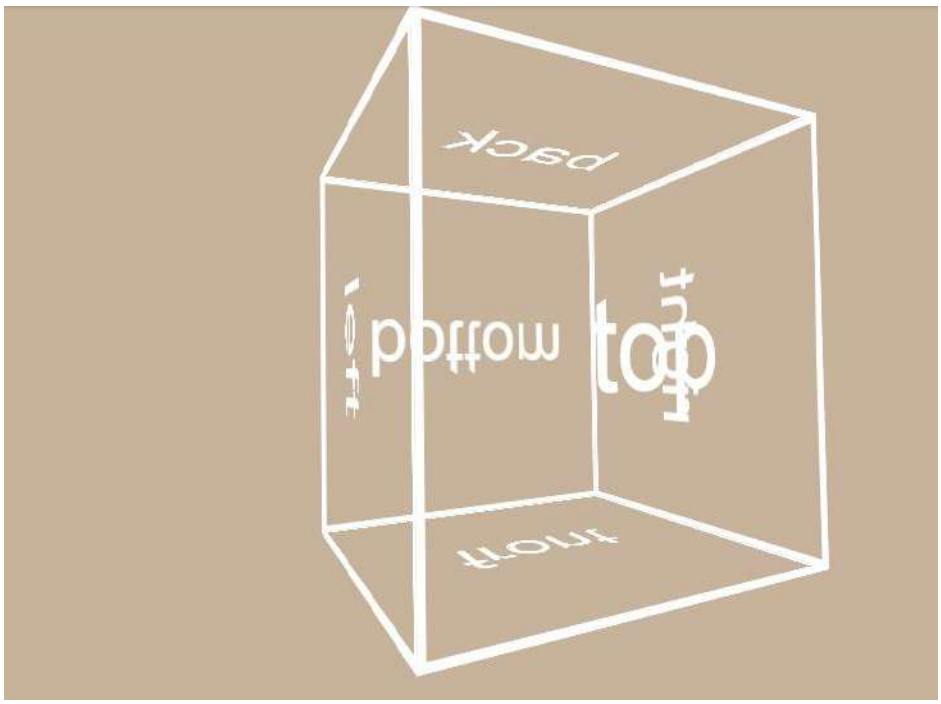


Рисунок 4.11 – Реалізація тривимірної css-анімації на прикладі куба

4.2.1. Створення крокової анімації

Крокові часові функції – step-start, step-end і steps – не описуються функціями Безьє і навіть не подаються кривими. Всі вони визначають стрибкоподібну зміну значень анімованих властивостей. Остання функція застосовується частіше інших, оскільки найкраще підходить для анімації персонажів і спрайтової анімації.

Часова функція steps () представляє анімацію у вигляді набору послідовно змінюючих один одного рівновіддалених станів. Вона має лише два аргументи: кількість кроків і одну або кілька точок зміни стану.

Кількість кроків ступінчастої анімації встановлюється першим аргументом крокової часової функції, яку представляють виключно позитивним цілочисельним значенням. Наприклад, односекундна анімація, яка виконується за 5 кроків, передбачає, що кожен крок триватиме 200 мс. Таким чином, елемент буде перерисовуватись п’ять разів на секунду через інтервали 200 мс, забезпечуючи 20% -ву зміну значення властивості на кожному кроці.

Щоб зрозуміти, як виконується крокова анімація, візьміть в руки книгу з картинками типу фліпбук (tlipbook) і швидко перегорніть її. На кожній сторінці книги міститься певний малюнок, який змінюється малюнком наступної сторінки, подібно кадрам кіноплівки. Якщо зміни в малюнках незначні, а самі вони змінюються дуже швидко, то анімація виглядатиме плавною, як у мультфільмі. Для отримання такого роду анімації досить скористатися спрайтами, властивістю background-posi tion і часовою функцією step.

На рис. 4.12 показаний спрайт, що містить зображення, швидка зміна яких в документі призводить до створення анімаційного зображення.

Тут під спрайтом розуміється колекція всіх видозмінених зображень, які беруть участь в анімації. Отже, кожне з зображень спрайту є окремим кадром анімації.

Згодом спрайт призначається як фонове зображення спеціально створеного контейнера елемента, розмір якого в точності збігається з розміром окремого зображення (кадру) спрайту. Візуалізація кадрів (зображень спрайту) виконується внаслідок анімації властивості background-posi tion за допомогою часової функції steps (). В даному випадку кількість кроків анімації повторює кількість зображень, включених у спрайт. Нескладно підрахувати, що число кроків анімації визначає кількість зупинок, які відбуваються в процесі «прокручування» фонового зображення в контейнері елемента.

Рисунок 4.12 – Ресурс спрайт-анімації

Спрайт, показаний на рис. 4.12, містить 22 зображення розміром 56x100 пікселів. Загальний розмір спрайту дорівнює 1232x100 пікселів. З використанням спрайту як фона елемента його початкове положення описується властивістю background-position зі значенням top left, рівнозначним значенню O o. Таке вихідне положення фонового зображення більш ніж виправдано: в браузерах, що не підтримують CSS-анімацію, фон елемента представлятиметься першим зображенням спрайту.

```
. dancer {
height: 100px;
width: 56px;
background-irnage: url (.. /irnages/dancer.png);
```

Ключовий момент у спрайтовій анімації полягає в застосуванні функції steps () для зміни значення властивості background-posi tion, що призводить до одночасної візуалізації в контейнері елемента тільки окремої частини спрайту.

У даному випадку фонове зображення не прокручується плавно, а виводиться стрибкоподібно за вказану кількість кроків.

Таким чином, наша спрайтова анімація зводиться до крокового зміщення спрайту, значна частина якого знаходиться поза областю перегляду, вліво.

Оскільки його ширина дорівнює 1232 пикселям, анімація елемента полягатиме в зміні його властивості background-position зі значення о о на О -1232px. При цьому область перегляду задається елементом div розміром лише 56x100 пікселів. Передача властивості background-posi tion значення О -1232px призводить до зміщення спрайту в крайнє ліве положення і повного видалення його з області перегляду. За відсутності повторення уздовж осі X таке налаштування позбавлює елемент div фонового зображення (ключовий кадр 100%). Всіляко уникайте використання властивості background-repeat під час налаштування анімації.

В кінцевому вигляді розглянута вище спрайтова анімація встановлюється таким кодом.

```
@keyfrarnes dance_in_place {
from {
background-position: 0 0;
to
background-position: -1232px Про;
.dancer {
background-image: url (.. /images/dancer.png);
animation-name: dance in place;
animation-duration: 4s;
animation-timing-function: steps (22, end);
animation-iteration-count: infinite;
```

Даний приклад покликаний показати, як за допомогою найпростіших методів можна додати в документ вельми складну анімацію. Для її створення достатньо покроково зміщувати спрайт зображень, доданий в одному з напрямків. У даному випадку анімація виконується за рахунок зміщення фонового зображення елементу.

Як відомо, перший параметр крокової часової функції встановлює кількість кроків анімації, а другий аргумент є ключовим словом start або end. Він вказує час внесення змін до анімованої властивісті: на початку або в кінці часового інтервалу кроку. За замовчуванням застосовується значення end, тому властивість background-posi tion буде змінювати своє значення в кінці кожного наступного інтервалу тривалістю 200 мс. При цьому перша зміна здійснюватиметься тільки після закінчення перших 200 мс анімації. Якщо уявити другий аргумент крокової часової функції значенням start, то зміна анімованої властивості виконуватиметься на початку часового інтервалу кожного кроку. Таким чином, анімація буде візуально виявлятися відразу ж,

з моменту її безпосереднього початку. Відмінності в значеннях start і end наочно проілюстровані на рис. 4.12.

```
@keyframes grayfade {
from {background-color: # BBB;}
to {background-color: # 333;}
.quickfader {animation: grayfade ls steps (S, start) forwards;}
.slowfader {animation: grayfade ls steps (S, end) forwards;}
```

Колір фону, який призначається елементу на кожному кроці анімації, показаний через заливку областей стовпчастої діаграми. Як бачимо, в разі застосування значення end колір фону для кожного кроку вибирається в його початковій часовій точці. Така поведінка пов'язана з тим, що перша зміна кольору здійснюється тільки в кінці першого (початку другого) кроку.

Для ключового слова start спостерігається зовсім інша ситуація: перша зміна кольору виконується на початку першого інтервалу, обумовлюючи установку фону елемента на початку, а не в кінці кожного наступного кроку анімації. В результаті зміна кольору фону виконується «наперед» – у варіанті end фоновий колір на другому кроці анімації відповідає такому на першому кроці варіанту start.

Така сама поведінка властива останньому етапу анімації. У варіанті start фоновий колір п'ятого кроку встановлюється в його початку і повною мірою відповідає кінцевому фоновому кольору елемента. Фоновий колір п'ятого кроку у варіанті end визначається відтінком, заданим в кінці четвертого кроку, а кінцевий фоновий колір елемент набуває в самому кінці анімації.

Найчастіше підбір значення аргумента, що визначає місце зміни анімованої властивості, викликає непереборні труднощі. Для більш чіткого визначення призначення кожного з ключових слів достатньо розуміти, що у варіанті start анімація позбавляється початкового (0%) ключового кадра, оскільки перша зміна кольору виконується на самому початку анімації, а у варіанті end вона позбавляється кінцевого (100%) ключового кадра.

Щоб запобігти скиданню фонового кольору в початкове значення по закінченні анімації, її налаштування потрібно завершити оголошенням ключового слова forwards, детально описаним у розділі «Режими анімації».

Значення step-start, як і часова функція steps (1, start), визначає однокрокову анімацію з єдиним ключовим кадром, що встановлюються в часовій точці 100%. Подібним чином значення step-end повністю рівнозначно функції steps (1, end) і визначає однокрокову анімацію з єдиним ключовим кадром, що встановлюється в часовій точці 0%.

Повернувшись до прикладу спрайтової анімації, давайте дещо ускладнімо танцювальні рухи нашого персонажа, забезпечивши його можливістю повороту вліво і вправо. Для цього нам знадобиться додати до елементу ще один тип анімації.

```
@keyframes move_around (
0%, 100% {
transform: translate (O, -40px) scale (0.9);
}
25% {
transform: translate (40px, 0) scale (l);
50% (
transform: translate (O, 40px) scale (ll); 75% (
transform: translate (-40px, 0) scale (l);
```

У наведеному вище коді створюється анімація move_around, яка призначається елементу поряд з уже наявною: її назва вказується в списку анімацій другою і відділяється від імені першої анімації комою.

```
файл sprite2.html).
.dancer (
background-image: url (.. /images/dancer.png);
animation-name: dance_in_place, move around;
animation-duration: 4s, lős;
animation-timing-function: steps (22, end), steps (5, end);
animation-iteration-count: infinite;
```

Зверніть увагу на те, що двома значеннями тепер забезпечуються всі властивості налаштування анімації, за винятком animation-iteration-count. Як ви пам’ятаєте, відсутність у властивості налаштування анімації такої самої кількості значень, як у властивості animation-name, передбачає їх дублювання до необхідної кількості. Оскільки нескінченно довго мають виконуватися обидва типи анімації, ключове infinite досить вказати лише один раз: воно буде застосовано до кожної іменованої анімації списку. Браузер самостійно збільшить кількість примірників значення властивості animation-iteration-count (у даному випадку infinite) до кількості елементів у списку назв анімацій.

```
<! DOCTYPE HTML>
<html>
<head>
<title> </ title>
<meta charset = "utf-8">
<link type = "text / css" href = "../ styles / style-edit.css" rel = "stylesheet">
```

```
<style type = "text / css" contenteditable id = "editme">. dancer {width: 56px;
height: 100px;
 background-image: url (c / dancer.png);
 animation-name: dance_in_place;
 animation-duration: 4s;
 animation-timing-function: steps (22, end);
 animation-iteration-count: infinite;
}
@keyframes dance_in_place {
 from {background-position: 0 0;}
 to {background-position: -1232px 0;}
}
</ Style>
<Style type = "text / css">
div {
        margin-left: 45%;
        margin-bottom: 1em;
}
.strip {border: 1px solid; width: 56px; height: 100px;}
.strip: hover {overflow: hidden;}
.strip img {
        animation: filmstrip 4s steps (22, end) infinite;
        position: relative;}
@keyframes filmstrip {
 from {left: 0;}
 to {left: -1232px;}
}
</ style>
</ head>
<body>
<div class = "dancer"> </ div>
<div class = "strip"> <img src = "c / dancer.png" alt = ""> </ div>
<p>
наведіть курсор миші на поле, щоб приховати зображення за межами поля
</ p>
</ body>
</ html>
```

Зміст файлу style-edit.css

```
head {
        display: block;
        }
style # editme {
        display: block;
        -webkit-user-modify: read-write-plaintext-only;
        white-space: pre-wrap;
```

```
        padding: 1ch 2ch; margin: 2ch;
        border: 0.25ch solid rgba (0,0,0,0.33);
        border-radius: 1ch;
        background-color: hsl (120,20%, 90%);
        font: 1.2rem Consolas, "Courier New", Courier, monospace;
        max-height: 50vh;
        overflow: auto;
        }
style # editme :: before {
        content: "Editable CSS";
        display: block;
        margin: -1ch -2ch 1em;
        padding: 0.75ch 1ch 0.25ch;
        background: rgba (0,42,0,0.15);
        color: # 666;
        border-bottom: 2px solid rgba (0,0,0,0.33);
        font-weight: bold;
        font-family: sans-serif;
        text-align: center;
}
body {
 font-family: Verdana, Geneva, sans-serif;
 font-size: 1.2em;
 padding: 1em;
}
```

4.3. Контрольні запитання та завдання

1. Поясніть правила CSS: @keyframes rotate, @keyframes Zindex.

2. Як змінювати швидкість обертання властивостями css?

3. Як змінювати затримку обертання властивостями css ?

4. Поясніть зміст 'animation-iteration-count: infinite'.

5. Які дії потрібно виконати для переміщення обертального диску по розміру екрану (рис. 4.1)?

6. Які дії потрібно виконати для переміщення обертального диску (рис. 4.1) по діагоналі екрану?

7. Як змінити розмір обертального диску (рис. 4.1) у різні моменти обертання?

8. Пояснити значення фрагмента коду з прикладу, результат якого поданий на рис. 4.7:

```
var foo=document.queryselector ( 'div');
foo.addeventlistener ( 'transitionend', function (e) {
```

console.log (e.propertyname);
console.log (e.elapsedtime);
})

9. Як створюється стиль (рисунок) зворотнього боку гральних карт з рисунку 4.8?

10. Які зміни потрібно ввести в код, наведений на рис. 4.9, щоб змінити колір, розмір, час падіння м'яча?

4.4. Завдання для самостійного розв'язання

1. Використовуючи код, наведений у табл. 4.2, добитися зображення в браузері, яке відповідає рис. 4.6. Досягніть зміни кольору тильного боку першої карти зліва. Змініть розмір карт, швидкість їх обертання з наведенням мишки, час зміни вертикального положення.

2. Додати зміни в код, щоб змінився вміст його файла в браузері та з'явилася четверта карта пікової масті.

3. Використовуючи код, який наведено в таблиці 4.4, вбудувати у файл * html і домогтися в браузері зображення, аналогічне рис. 4.11. Визначити, яким символам клавіатури відповідають клавіші 36, 37, 38, 39 і перевірити можливість обертання куба у 3d-просторі.

Змінити лінійні розміри куба, колір граней, фону, на довільні. На будь-яку з граней куба помістити довільне зображення, наприклад фотографію у форматі *.jpg, адаптувавши під розмір межі.

Змініть колір межі, що відповідає назві «права».

4. Створіть *Html-файл із вмістом з табл. 4.4. Перевірте його працездатність в браузері. Внесіть в нього зміни для розташування м'яча в центрі екрану. Створіть м'яч, який би падав та обертався з довільною швидкістю. Досягніть, щоб падіння і відскоки м'яча були безперервні після завантаження файла.

5. Створіть анімацію, аналогічну тій, що реалізована на прикладі рис. 4.12.

5. СТВОРЕННЯ АНІМАЦІЙНИХ ЕФЕКТІВ ЗА ДОПОМОГОЮ jQuery

5.1. Стислий огляд особливостей бібліотеки

Бібліотека jQuery містить кілька крос-браузерних методів для анімації елементів, наприклад, ковзання і плавне зникнення, без залучення додаткових бібліотек або плагінів. Для розширення можливостей роботи з анімацією скористайтеся бібліотекою jQuery UI (http://jqueryui.com), яка містить набір інтерфейсних взаємодій, ефекти, віджети і теми.

CSS-стилі надають елементам сторінки візуальні властивості, які описують їх зовнішній вигляд. jQuery-анімація являє собою інтерактивний процес зміни властивостей html-елементів від одного значення до іншого.

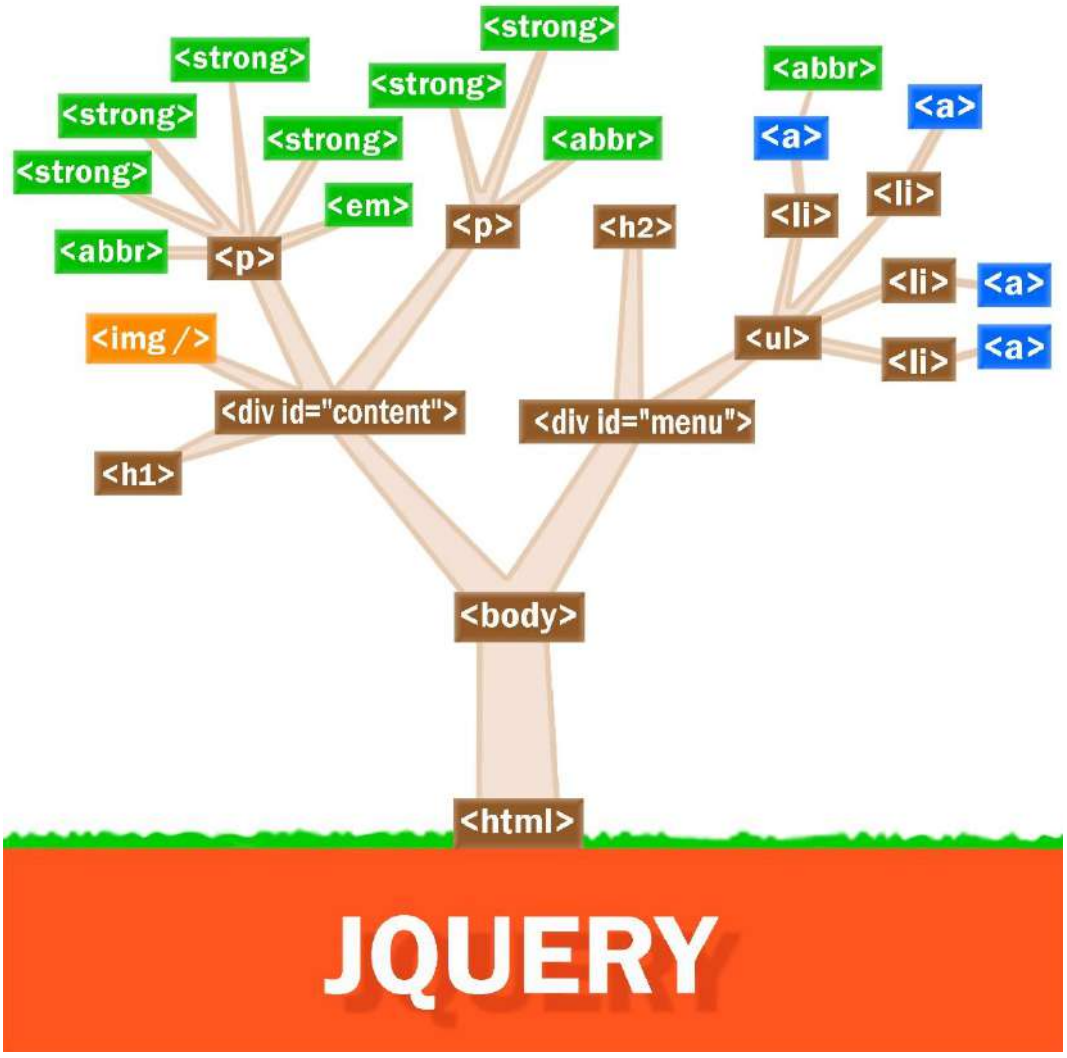


Рисунок 5.1 – Ієрархія бібліотеки jqwery

Динамічна зміна елементів веб-сторінок

Бібліотека jQuery спрощує процес відбору елементів HTML-сторінок. За допомогою методів jQuery відбуваються маніпуляції з об’єктною моделлю документа DOM. Щоб відібрати групу елементів, потрібно передати селектор функції jQuery. Як селектор елемента може виступати сам елемент, його ідентифікатор або клас, а також комбінація селекторів:

$ ("a")
$ ("# some-id")
$ (". someclass")
$ ("header> ul: has (a)")

Функція $ () повертає об'єкт jQuery, що містить масив елементів DOM – так званий обгорнутий набір, відповідний вказаним селектору. Більшість методів повертає по завершенні дій початковий набір елементів.

Створення власних ефектів за допомогою методу .animate ()

Ефекти, яких немає в бібліотеці jQuery, можна створювати за допомогою методу animate (). Інтерпретатор браузера динамічно, без перезавантаження сторінки, змінює вибрані властивості на зазначені значення. Анімація відбувається для всіх елементів обернутого набору. Щоб додати ефекти для конкретного елемента, потрібно скористатися фільтрами jQuery для відбору.

Метод дозволяє анімувати css-властивість, що має **числове значення**, наприклад, font-size, opacity, border-width, margin, padding, height, width, background-position і т.д. При цьому імена властивостей мають бути вказані разом – fontSize, paddingLeft, або має використовуватися css-еквівалент властивості – font-size. Числові значення властивостей не вкладаються в лапки.

Для будь-якої властивості попередньо має бути встановлено початкове значення, а в css-оголошенні має використовуватися повний запис кожної властивості, тобто замість властивості border мають бути задані значення для border-style, border-width і т.д.

Функція зворотного виклику викликається один раз після завершення анімації. Функції не передається жодних аргументів, але анімації виконуються для елемента, переданого властивості this як контекст.

Значеннями властивостей можуть також виступати hide, show або toggle, в результаті чого до елементу застосовується обчислюване значення – відображення, приховування або перемикання вихідних станів властивостей.

Метод .animate () дозволяє змінювати css-властивості обраних елементів з можливістю одночасної анімації декількох властивостей, задаючи тривалість анімації в мілісекундах.

$ ("div"). animate ({left: "200px", top: "200px"}, 500);

Дана анімація водночас застосовується до властивості left, для якої задано значення 200px, і до властивості top зі значенням 200px, тривалість анімації задана 500ms.

Для елемента можна задавати відносне переміщення з кожним викликом анімації за допомогою операторів + =, - =, * =, / =, наприклад, $ ("div"). animate ({left: "+ = 200", top: "- = 200 "}, 500).

Метод .**animate** () має дві форми запису. У першій методу передаються чотири аргументи:

– **.animate**({властивість: «значення1», властивість2: «значення2»}, тривалість, функція переходу, function () {...});

– **{властивість1: «значення1**»} – об’єкт властивостей, які збираємося анімувати у форматі властивість: «значення», перераховані в фігурних дужках через кому;

– **тривалість** – необов’язковий параметр тривалості анімації, що задає час в мілісекундах або за допомогою ключових слів slow, fast, normal;

– **функція переходу** – необов’язкове ім’я функції переходу;

– **function** () {...} – необов’язкова функція зворотного виклику.

Друга форма приймає два аргументи – об’єкт властивостей і об’єкт додаткових функціональних можливостей:

– **.animate** ({властивість: «значення1», властивість2: «значення2»}, {duration:«значення», easing:«значення», specialEasing: {

– **властивість**: «easing1», властивість2: «easing2»}, complete: function () {...}, queue: true, step: callback});

– **{властивість1: «значення1»}** – об’єкт властивостей, які збираємося анімувати в форматі властивість: «значення», перераховані в фігурних дужках через кому.

Duration – число або ключове слово, значення за замовчуванням 400. Встановлює тривалість анімації.

Easing – ім’я функції переходу. Значення за замовчуванням swing – коливальний перехід, друге доступне значення – linear – лінійний перехід. Розширені можливості реалізуються за допомогою плагінів, дивіться jQuery UI.

Queue – поміщає анімацію в чергу ефектів, за замовчуванням true, значення false – негайно відтворює анімацію.

SpecialEasing – об’єкт, що містить одну і більше властивостей і їх значень, що описують функції переходу.

Step – функція, яка викликатиметься після закінчення кожного етапу анімації і застосовуватиметься для кожного елемента обернутого набору. Функції передається порядковий номер етапу і внутрішній об’єкт, що описує ефект.

Progress – функція, яка викликатиметься після кожного кроку анімації, тільки один раз за анімацію незалежно від кількості анімаційних властивостей.

Complete – функція, яка має бути викликана після закінчення відтворення анімації.

Start – функція, яка буде викликана на початку анімації.

Done – функція, викликана по завершенні анімації.

Fail – функція, що викликається в разі неможливості завершення анімації.

Always – функція, яка буде викликана в разі зупинки або неповного завершення анімації.

5.2. Анімаційні ефекти jQuery

5.2.1. Метод .fadeIn ()

Метод управляє прозорістю, показуючи прихований елемент, при цьому властивість opacity обраного елемента змінюється від 0 до 1. Для цього на сторінці з’являється необхідний простір для елемента, при цьому інші елементи можуть зрушити з місця.

Синтаксис – ***.fadeIn (тривалість, функція по завершенні анімації)***

Тривалість – необов’язковий параметр, задає швидкість прояву ефекту за допомогою ключових слів «fast», «normal», «slow» або числових значень. За замовчуванням використовується значення «normal», рівне 400 мілісекунд.

Функція по завершенні анімації – необов’язковий параметр, задає функцію, яка буде викликана після прояву елемента.

Параметр fadeIn (об’єкт властивостей) – являє собою пари додаткових опцій, зазначених у форматі властивість: «значення».

Параметр fadeIn (тривалість, функція переходу, функція по завершенні анімації) тривалість – необов’язковий параметр, задає швидкість прояву ефекту.

Функція переходу – необов’язковий параметр, задає функцію переходу. Значення за замовчуванням swing – коливальний перехід, друге доступне значення – linear – лінійний перехід.

5.2.2. Метод .fadeOut ()

Метод змушує елемент зникнути, зробивши його прозорим, зберігаючи на сторінці місце, займане елементом. Властивість opacity обраного елемента змінюється від 1 до 0.

Синтаксис – ***.fadeOut (тривалість, функція по завершенні анімації)*** (тривалість – необов'язковий параметр, задає швидкість прояву ефекту за допомогою ключових слів «fast», «normal», «slow» або числових значень). Значення за замовчуванням – 400 мілісекунд.

Синтаксис – ***.fadeOut (об'єкт властивостей)***.

Об'єкт властивостей – являє собою пари додаткових опцій, вказаних у форматі властивість: «значення».

Викликається метод з параметрами – ***.fadeOut*** (тривалість, функція переходу, функція по завершенні анімації).

Тривалість – необов'язковий параметр, задає швидкість прояву ефекту.

Функція переходу – необов'язковий параметр, задає функцію переходу.

Функція по завершенні анімації – необов'язковий параметр, задає функцію, яка буде викликана після прояву елемента.

5.2.3. Метод .fadeTo ()

Метод дозволяє змінити ступінь прозорості до заданого значення, заходів, $ («element»). FadeTo («normal»,. 50) ;.

Синтаксис – ***.fadeTo (тривалість, прозорість, функція по завершенні анімації)***

Тривалість – задає швидкість прояву ефекту.

Прозорість – число від 0 до 1, що задає прозорість елемента.

Функція по завершенні анімації – необов'язковий параметр, описує функцію, яка буде викликана після прояву елемента.

Синтаксис – ***fadeTo (тривалість, прозорість, функція переходу, функція по завершенні анімації)***

Тривалість – задає швидкість прояву ефекту.

Прозорість – число від 0 до 1, що задає прозорість елемента.

Функція переходу – необов'язковий параметр, задає функцію переходу.

Функція по завершенні анімації – необов'язковий параметр, описує функцію, яка буде викликана після прояву елемента.

5.2.4. Метод .fadeToggle ()

Якщо елемент прихований, то він з'являється на екрані, якщо видно, то зникає.

Синтаксис **– *.fadeToggle (тривалість, функція переходу, функція по завершенії анімації).***

Тривалість – задає швидкість прояву ефекту.

Функція по завершенні анімації – необов'язковий параметр, описує функцію, яка буде викликана після прояву елемента.

Синтаксис – ***.fadeToggle*** (об'єкт властивостей).

Об'єкт властивостей – являє собою пари додаткових опцій, зазначених у форматі властивість: «значення».

5.2.5. Метод .hide ()

Метод приховує видимий елемент. При встановленні швидкості елемент зникає, мов би звужуючись. Щоб уповільнити дію, потрібно передати значення тривалості.

Синтаксис – ***.hide (). Метод вказується без параметрів.***

Синтаксис – ***.hide (тривалість, функція по завершенні анімації).***

Тривалість – необов'язковий параметр, задає швидкість прояву ефекту.

Функція по завершенні анімації – необов'язковий параметр, описує функцію, яка буде викликана після прояву елемента.

Об'єкт властивостей – являє собою пари додаткових опцій, зазначених в форматі властивість: «значення».

Синтаксис – ***.hide (тривалість, функція переходу, функція по завершенні анімації)***

Тривалість – задає швидкість прояву ефекту.

Функція переходу – необов'язковий параметр, задає функцію переходу.

Функція по завершенні анімації – необов'язковий параметр, описує функцію, яка буде викликана після прояву елемента.

5.2.6. Метод .show ()

Цей метод показує прихований раніше елемент. Якщо не задано значення швидкості, то елемент з'являється моментально, якщо швидкість задана, то елемент з'являється від верхнього лівого до нижнього лівого кута.

Метод ***show ()*** вказується без параметрів.

Синтаксис – ***.show (тривалість, функція по завершенні анімації)***

Тривалість – необов'язковий параметр, задає швидкість прояву ефекту.

Функція по завершенні анімації – необов'язковий параметр, описує функцію, яка буде викликана після прояву елемента.

Синтаксис – ***.show (об'єкт властивостей)***

Об'єкт властивостей – являє собою пари додаткових опцій, зазначених в форматі властивість: «значення».

Синтаксис – ***.show (тривалість, функція переходу, функція по завершенні анімації)***

Тривалість – задає швидкість прояву ефекту.

Функція переходу – необов'язковий параметр, задає функцію переходу.

Функція по завершенні анімації – необов'язковий параметр, описує функцію, яка буде викликана після прояву елемента.

5.2.7. Метод .toggle ()

Цей одиночний метод, перемикає вибраний пункт одного стану в інший, залежно від його поточного стану, приховуючи або відображаючи його.

Синтаксис – ***.toggle (тривалість, функція по завершенні анімації)***

Тривалість – необов'язковий параметр, задає швидкість прояву ефекту.

Функція по завершенні анімації – необов'язковий параметр, описує функцію, яка буде викликана після прояву елемента.

Синтаксис – ***.toggle (об'єкт властивостей)***

Об'єкт властивостей – являє собою пари додаткових опцій, зазначених у форматі властивість: «значення».

Синтаксис – ***.toggle (тривалість, функція переходу, функція по завершенні анімації)***

Тривалість – задає швидкість прояву ефекту.

Функція переходу – необов'язковий параметр, задає функцію переходу.

Функція по завершенні анімації – необов'язковий параметр, описує функцію, яка буде викликана після прояву елемента.

Синтаксис **– *.toggle (display)***

Display – якщо параметр display встановлений, то елемент буде показаний.

5.2.8. Метод .slideDown ()

Цей метод змушує прихований елемент з'явитися на веб-сторінці. Елемент проявляється поступово – спочатку його верхня частина, і в міру прояву іншої частини те, що знаходилося під елементом, зсувається вниз. Тому для того, щоб контент не переміщувався по сторінці, можна використовувати абсолютне позиціонування елемента {position: absolute;}. Якщо необхідно розмістити даний елемент щодо іншого, то задайте відносне позиціонування {position: relative;} для елемента, який оточує абсолютно позиційований елемент.

Синтаксис **– *.slideDown (тривалість, функція по завершенні анімації).***

Тривалість – необов'язковий параметр, задає швидкість прояву ефекту.

Функція по завершенні анімації – необов'язковий параметр, описує функцію, яка буде викликана після прояву елемента.

Синтаксис – ***.slideDown (об'єкт властивостей).***

Об'єкт властивостей – являє собою пари додаткових опцій, зазначених у форматі властивість: «значення».

Синтаксис – ***.slideDown (тривалість, функція переходу, функція по завершенні анімації).***

Тривалість – необов'язковий параметр, задає швидкість прояву ефекту.

Функція переходу – необов'язковий параметр, задає функцію переходу.

Функція по завершенні анімації – необов'язковий параметр, описує функцію, яка буде викликана після прояву елемента.

5.2.9. Метод .slideUp ()

Цей метод змінює властивість height елемента, поки він не стане рівним 0, після приховує елемент display: none ;. При цьому видалення починається знизу, і якщо для елемента не задано позиціонування, то контент, який перебував нижче, переміщується вгору.

Синтаксис – .slideUp (тривалість, функція по завершенні анімації).

Тривалість – необов'язковий параметр, задає швидкість прояву ефекту.

Функція по завершенні анімації – необов'язковий параметр, описує функцію, яка буде викликана після прояву елемента.

Синтаксис – ***.slideUp (об'єкт властивостей).***

Об'єкт властивостей – являє собою пари додаткових опцій, зазначених у форматі властивість: «значення».

Синтаксис – ***.slideUp (тривалість, функція переходу, функція по завершенні анімації).***

Тривалість – необов'язковий параметр, задає швидкість прояву ефекту.

Функція переходу – необов'язковий параметр, задає функцію переходу.

Функція по завершенні анімації – необов'язковий параметр, описує функцію, яка буде викликана після прояву елемента.

5.2.10. Метод .slideToggle ()

Даний метод приховує видимий елемент і показує прихований, тобто може використовуватися як перемикач, що дозволяє як відображати, так і приховувати елемент. При цьому елемент проявляється поступово, зверху вниз.

Синтаксис **– *.slideToggle (тривалість, функція по завершенні анімації)***

Тривалість – необов'язковий параметр, задає швидкість прояву ефекту.

Функція по завершенні анімації – необов'язковий параметр, описує функцію, яка буде викликана після прояву елемента.

Синтаксис – ***.slideToggle (об'єкт властивостей)***

Об'єкт властивостей – являє собою пари додаткових опцій, зазначених у форматі властивість: «значення».

Синтаксис – ***.slideToggle (тривалість, функція переходу, функція по завершенні анімації).***

Тривалість – необов'язковий параметр, задає швидкість прояву ефекту.

Функція переходу – необов'язковий параметр, задає функцію переходу.

Функція по завершенні анімації – необов'язковий параметр, описує функцію, яка буде викликана після прояву елемента.

5.2.11. Управління чергою анімації

Анімація елемента реалізується тільки за допомогою методів, що створюють анімаційні ефекти. Якщо анімація реалізована ланцюжком методів, кожен ефект виконується послідовно, один за одним, наприклад, $ («div»). slideDown (). fadeOut (); спочатку почнеться ковзання блоку вниз, а вицвітання покладене в чергу «fx» і буде викликано тільки по завершенні ковзання.

Щоб додати ефекти користувача, створюються функції, які також додаються в чергу «fx». Черга є масивом функцій, існуючих на рівні елемента, які зберігаються в jQuery.data. Кожен елемент може мати одну або кілька

черг функції, але зазвичай використовується тільки одна черга за замовчуванням «fx».

Функції включаються в чергу за допомогою методу .queue (), без нього функції виконуватися не будуть. Кожна функція по завершенні має викликати метод .dequeue (), щоб передати управління наступній функції у черзі.

```
$ ( "div"). slideUp ();
$ ( "div"). queue (function () {
document.write ( "Hellow");
$ (this) .dequeue ();
});
```

5.2.12. Метод .queue ()

Метод показує поточну чергу анімації або маніпулює чергою функцій, що створюють анімаційні ефекти, для елемента / елементів набору.

Ім’я черги – рядок, що містить ім'я черги. За замовчуванням поточна черга має зарезервоване ім’я «fx». Повертає масив функцій, що знаходяться в черзі.

Синтаксис – ***.queue (ім’я черги, функція)***

Ім’я черги – рядок, що містить ім’я черги.

Функція – функція зворотного виклику, яка додається в кінець черги з ім'ям черги для всіх елементів обернутого набору.

Синтаксис – ***.queue (ім’я черги, ім’я нової черги)***

Ім’я черги – рядок, що містить ім’я черги.

Ім’я нової черги – замінює поточну чергу всіх елементів набору на нову чергу, складається з масиву функцій.

5.2.13. Метод .dequeue ()

Даний метод ставить наступну анімацію в чергу. Це означає, що якщо виконується анімація, коли цей метод викликається, то нова анімація буде розпочата відразу після того, як поточна закінчить своє виконання. Виконує першу функцію черги для всіх елементів набору, після виконання функція видаляється з черги. Обробники події будуть виконувати і видаляти з черги по черзі наступні функції. Якщо метод викликається всередині функції, що додається в чергу, це викличе почергове виконання всіх функцій, що входять у чергу.

Синтаксис **– *.dequeue (ім’я черги)***

Ім’я черги – рядок, що містить ім’я черги.

5.2.14. Метод .clearQueue ()

Даний метод видаляє всі функції анімації з черги, не тільки поточну, але й усі наступні, без їх виконання. Може використовуватися для видалення будь-яких черг.

Синтаксис – ***.clearQueue (ім'я черги)***

Ім'я черги – рядок, що містить ім'я черги.

5.2.15. Метод .delay ()

Даний метод дозволяє встановити відстрочку для запуску ефектів, чекаючи певну кількість мілісекунд, перш ніж запустити наступний ефект у черзі.

Синтаксис – ***.delay (число або рядок, ім'я черги)***

Рядок – встановлює тривалість затримки в мілісекундах або за допомогою ключових слів – fast (еквівалентно 200ms) і slow (еквівалентно 600ms).

5.2.16. Метод .stop ()

Цей метод зупиняє поточну анімацію в черзі відразу після його запуску для елементів набору. Якщо при виклику методу передаються деякі аргументи, то ви також можете очистити чергу і визначити, чи повинні елементи при зупинці анімації залишитися на місці, або вони мають повернутися до початкового стану. За замовчуванням обидва параметри мають значення false.

Синтаксис – ***.stop (логічне значення1, логічне значення2):***

– логічні значення 1 – значення true зупиняє всі анімаційні ефекти, що знаходяться в черзі;

– логічні значення 2 – значення true зупиняє всі ефекти, крім поточного.

5.2.17. Метод .finish ()

Даний метод зупиняє поточну анімацію, видаляє всі черги анімації і завершує всі анімації для відповідних елементів.

Синтаксис – ***.finish (ім'я черги)***

Ім'я черги – задає ім'я черги анімації, яку потрібно зупинити.

За замовчуванням – черга «fx».

5.3. Управління анімацією через властивості об'єкта jQuery

Властивості дозволяють змінити швидкість виконання анімації. Як значення вказується кількість мілісекунд. Значення за замовчуванням 13ms. Зменшуючи швидкість, можна домогтися більш плавного виконання ефектів.

Синтаксис – jQuery.fx.interval = 500; Властивість jQuery.fx.off.

Використовується для того, щоб глобально відключити або підключити анімацію. Значення за замовчуванням false, що відповідає виконанню анімації. Якщо задано значення true, то всі методи анімації будуть відключені, а елементи повернуться до свого початкового стану.

Використовуючи функції jQuery, можна анімувати елементи веб-сторінки. Всі ефекти jQuery грунтуються на зміні CSS властивостей. Можна використовувати вже готові ефекти, такі як приховування, поява, рух елементів або створювати свої більш складні, засновані на зміні декількох властивостей елемента.

У цьому розділі розглядаються функції, що приховують елементи за допомогою зміни видимості, розмірів і прозорості.

5.3.1. Поява і приховування елемента

Функція show () дозволяє показувати прихований елемент. Елемент можна показати миттєво, для цього слід використовувати такий запис: $ («div»). show ().

Функція .hide () дозволяє приховувати елемент. Елемент можна приховати миттєво, для цього слід використовувати такий запис:

$ ("div"). hide ().

У процесі приховування елемента йому присвоюється властивість display із значенням none, з появою повертається вихідне значення властивості display.

Використовуючи додаткові значення цих функцій, можна плавно приховувати / показувати елемент, для цього слід додати параметр тривалості дії анімації (duration). У наступному прикладі показано, як встановити швидкість приховування 400 мілісекунд.

5.3.2. Згортання і розгортання елемента

Для плавного згортання/розгортання елемента використовуються функції .slideDown () і .slideUp (). Ці функції схожі на попередні функції приховування / показу елементів і володіють тими самими параметрами.

$ ("div"). slideDown (400, function () {})
$ ("div"). slideUp (400, 'linear', function () {})

У процесі приховування елемента йому присвоюється властивість display із значенням none, з появою повертається вихідне значення властивості display.

Почергове розгортання і згортання елементів

Функція .slideToggle () послідовно згортає елемент або, якщо елемент згорнуть, розгортає його. Ця функція має параметри, якими володіють попередні функції

$ ("div"). fadein (400, function () {})
$ ("div"). slideToggle (400, 'linear', function () {}).

У процесі приховування елемента йому присвоюється властивість display із значенням none, з появою повертається вихідне значення властивості display.

Зміна рівня прозорості елемента Функція .fadeTo () дозволяє змінювати прозорість елемента до заданого рівня. Крім основних параметрів (тривалості дії анімації (duration), запуску функції по завершенні анімації і параметра easing) для цієї функції доданий параметр opacity, який визначає ступінь прозорості елемента від 1 (не прозорий) до 0 (прозорий).

$ ("div"). fadeTo (400, 0.4)

5.3.3. Почергова поява та приховування елементів

Функція fadeToggle () дозволяє послідовно показувати і приховувати елемент, так як це роблять методи .fadeIn () і .fadeOut (), тобто плавно змінюючи прозорість. Ця функція підтримує ті самі параметри, що і попередні функції.

$ ("div"). fadeToggle (400, function () {})

З приховуванням елементуа йому присвоюється властивість display із значенням none, з появою повертається вихідне значення властивості display.

5.3.4. Створення користувальницької анімації

Призначена для користувача анімація створюється за допомогою функції .animate () і здійснюється за рахунок зміни зазначених властивостей CSS-елемента.

Функція **.animate ()** має такі необов'язкові параметри (знайомі нам за попередніми функціями анімації): тривалість дії анімації (duration), запуск функції по завершенні анімації і параметр easing. Додатковий параметр, який

додається першим, визначає змінні css-властивості, називається properties. У наступному прикладі показаний синтаксис функції .animate ().

$ ("div"). animate (properties, duration, easing, function)

Параметр properties визначає кінцеві значення змінюваних css-властивостей, тобто якщо висота елемента за замовчуванням 500px, а як значення параметра properties ви вкажете 200px, то розмір елемента зміниться з 500px до 200px.

Функція animate () може змінювати тільки CSS-властивості, значеннями яких є числа, рядкові значення або шістнадцяткові значення змінені бути не можуть. Крім абсолютних одиниць можуть застосовуватися відносні значення, наприклад, width: "-= 100", що змінить ширину елемента на 100px. Крім того, можна задавати стандартні значення hide, show, toggle, які приховають, покажуть або змінять видимість елемента на протилежний.

$ ('div'). animate ({width: "+ = 50", height: "200px"}, 5000);

Можна послідовно виконувати кілька анімаційних ефектів, заданих як функцією animate, так і іншими функціями анімації.

У наступному прикладі до елемента з класом animeConsecutivelyEl послідовно застосовується дві функції animate () і функція fadeTo ().

Тобто в цьому прикладі показано чергу з трьох функцій.

$ (".animeConsecutivelyEl"). animate ({width: "-= 50", borderWidth:"7px"}, 5000)
.animate ({height: "200px"}, 5000) .fadeTo (400,0.2).

Зверніть увагу на відмінності між послідовною та одночасною анімаціями.

Розглянемо далі зміну черги анімації

Збільшуючи послідовно анімації до об’єкта, ми створюємо чергу анімації, яка буде виконуватися незалежно від проміжних функцій.

Розглянемо приклад.

```
$ ( '# animeConsecutively'). click (function ()
 {
 $ ( ". animeConsecutivelyEl"). animate ({width: "-= 50 ",
borderWidth: "7px"}, 5000);
 $ ( ". animeConsecutivelyEl"). animate ({height: "200px"}, 5000);
 $ ( ". animeConsecutivelyEl"). css ( "background-color ", 'red');
 $ ( ". animeConsecutivelyEl"). fadeTo (400,0.2);
 });
```

Для елемента з класом animeConsecutivelyEl призначено чергу з різних функцій і три з них функції анімації, а одна змінює фоновий колір. Під час виконання даного коду три функції анімації будуть поміщені в свою чергу і виконуватимуться послідовно, а функція зміни фонового кольору буде поміщена в свою чергу і виконається водночас з першою функцією animate ().

Для того щоб послідовно виконувати не тільки функції анімації, але й інші функції, використовується метод .queue ().

Змінимо попередній приклад і визначимо в чергу зміну фонового кольору.

```
$('#animeConsecutively').click(function()
{ $(".animeConsecutivelyEl").animate({ width:"-=50", borderWidth:"7px"},5000);
   $(".animeConsecutivelyEl").animate({ height: "200px"},5000);
   $(".animeConsecutivelyEl").queue(function () {
         $(this).css("background-color",'red');
        $(this).dequeue();
   });
   $(".animeConsecutivelyEl").fadeTo(400,0.2);
   });
```

Там, де необхідно в чергу анімації вставити зміну фонового кольору, додаємо функцію queue (), всередині якої визначаємо функцію з необхідною зміною фону. Тепер після другої анімації буде змінений колір фону. Для того, щоб вийти зі вставленої функції і повернутися в чергу анімації, слід додати метод .dequeue () або для функції, що викликається, додати параметр next (.queue (function (next) {}).

Розглянемо очищення черги функцій

Для очищення черги функцій у зазначених елементах використовується метод .clearQueue (). У наступному прикладі виконується тільки перша анімація, всі наступні, що стоять в черзі, будуть видалені.

```
$ ( ". anime Consecutively El")
.animate ({width: "-= 50 ", borderWidth:" 7px "}, 5000);
.animate ({height: "200px"}, 5000)
.fadeTo (400,0.2);
$ ( ". animeConsecutivelyEl"). clearQueue ();
```

Розглянемо припинення анімації

За замовчуванням наступна в черзі анімація починається відразу після завершення попередньої. За допомогою методу .delay () можна встановити

затримку в мілісекундах між анімаціями. У наступному прикладі між анімаціями встановлений проміжок дорівнює одній секунді.

```
$ ( ". animeConsecutivelyEl")
.animate ({width: "-= 50 ", borderWidth:" 7px "}, 5000);
.delay (1000)
.animate ({height: "200px"}, 5000).
```

Розглянемо завершення і зупинку анімації

Для миттєвого завершення анімації використовується метод .finish. під час використання цього методу анімація миттєво переходить до кінцевого стану, зазначеного в налаштуваннях анімації. Метод .finish () з'явився у версії 1.9 і не працює в попередніх версіях.

Для зупинки анімації в даний момент використовується метод .stop (). Цей метод має кілька способів використання залежно від наявності та кількості встановлених параметрів.

У наступному прикладі метод stop () припинить дію поточної анімації і перейде до наступної:

```
$ ( ". animeEl"). stop ().
```

Для зупинки всієї черги анімації, слід встановити для параметра queue значення true, як показано у даному прикладі:

```
$ ( ". animeEl"). stop (true).
```

Параметр jumpToEnd визначає, залишатися елементу в тому стані, при якому був запущений метод stop (), або прийняти стан, заданий за замовчуванням для даної анімації, як показано у такому прикладі

```
$ ( ". animeEl"). stop (true, true).
```

Розглянемо зміну швидкості анімації

За замовчуванням швидкість анімації jQuery дорівнює 13 мілісекунд, але це значення можна змінити, використовуючи властивість $ .fx.interval. Для вказівки часового проміжку між кадрами анімації слід вказати значення цього $ .fx.interval = 500.

У властивості $ .fx.interval fx є назвою черги, прийнятої за замовчуванням jQuery.

Для скасування всіх анімацій на сторінці використовується властивість jQuery.fx.off з встановленим значенням true, відповідно для запуску

зупиненої анімації значення властивості jQuery.fx.off слід встановити, як false jQuery.fx.off = true.

Вибір анімованого елемента

Для вибору анімованого в даний момент елемента використовується селектор: animated.

У наступному прикладі для div-елемента з діючої анімації буде змінений фоновий колір $ («div: animated») .css («bacgroundColor», «red»);

Таблиця 5.1 – Обробник події hover

Метод	Опис
.hover ()	Встановлює обробник(и) двох подій: mouseenter і mouseleave: .hover (handlerIn (eventObject), handlerOut (eventObject)): jQueryv: 1.0 Встановлює функції handlerIn і handlerOut як обробників подій mouseenter і mouseleave, на вибрані елементи сторінки .handlerIn (eventObject), handlerIn (eventObject) – функції, що будуть встановлені як оброблювачі. З викликом вони отримуватимуть об'єкт події eventObject. hover (handlerInOut (eventObject)): jQueryv: 1.4 Встановлює процедуру handlerInOut як обробник обох подій (mouseenter і mouseleave) для обраних елементів сторінки. handlerInOut (eventObject) – функція, що буде встановлена як обробник. З викликом вона отримуватиме об'єкт події eventObject. Прибрати встановлені обробники можна за допомогою методу unbind ().
Приклад	***// встановимо обробники елемента з ідентифікатором foo. Обробники будуть виводити текстові повідомлення $ ('# foo'). hover (function () { alert ('Ви потрапили на територію елемента "foo", відому своєю валидною версткою' + 'і наявністю диких обробників подій.'); }, function () { alert ('Ви покинули територію елемента "foo". Ми будемо раді бачити вас знову.'); });***

Таблиця 5.2 – Методи управління анімацією

Метод	Опис
.animate ()	Виконує анімацію, яка була створена користувачем.
.queue ()	Надає/змінює (залежно від параметрів) чергу функцій.
.clearQueue ()	Очищує чергу функцій.
.dequeue ()	Починає виконання наступної функції в черзі.
.stop ()	Зупиняє виконання поточної анімації.
.delay ()	Призупиняє виконання наступних анімацій на заданий час.
.toggle ()	По черзі виконує виклик однієї з декількох заданих функцій.
jQuery.fx.interval	Містить часовий проміжок між кадрами анімації.
jQuery.fx.off	Скасовує виконання всіх анімацій.
.animate ()	Виконує анімацію, яка була створена користувачем.
.queue ()	Надає/змінює (залежно від параметрів) чергу функцій.

Таблиця 5.3 – Методи стандартної анімації

Метод	Опис
.hide () ***.show ()***	Приховує/показує елементи на сторінці (за рахунок плавної зміни його розміру й прозорості).
.slideUp () ***.slideDown ()***	Розгортає/згортає елементи на сторінці (за рахунок плавної зміни висоти елементів).
slideToggle ()	По черзі розгортає/згортає елементи на сторінці, як це роблять .slideUp () і .slideDown ().
.fadeIn () ***.fadeOut ()***	Приховує/показує елементи на сторінці за рахунок плавної зміни прозорості.
.fadeTo ()	Плавно змінює прозорість елементів.
.fadeToggle ()	По черзі приховує/показує елементи на сторінці, як це роблять .fadeIn () і .fadeOut ().

Метод ***.animate ()$ (Selector) .animate ({params}, speed, callback);*** виконує задану користувачем анімацію, з обраними елементами. Анімація відбувається за рахунок плавної зміни CSS-властивостей у елементів. Функція має два варіанти використання:

.animate (properties, [duration], [easing], [callback]).

Properties – список CSS-властивостей, які беруть участь в анімації і їх кінцевих значень;

Duration – тривалість виконання анімації. Може бути задана в мілісекундах або рядковим значенням 'fast' або 'slow' (200 і 600 мілісекунд); easing – зміна швидкості анімації (чи буде вона сповільнюватися до кінця

виконання або навпаки прискориться); callback – функція, яка буде викликана після завершення анімації;

.animate (properties, options);

– properties – список CSS-властивостей, які беруть участь в анімації і їх кінцевих значень. (див. опис в додатках B, D); Додаткові опції. Мають бути представлені об’єктом, у форматі опція: значення. Варіанти опцій:

– *duration* – тривалість виконання анімації

– *easing* – зміна швидкості анімації (чи буде вона сповільнюватися до кінця виконання або навпаки прискоритися).

– *complete* – функція, яка буде викликана після завершення анімації.

– *step* – функція, яка буде викликана після кожного кроку анімації.

– *queue* – логічне значення, яке вказує, чи слід поміщати поточну анімацію в чергу функцій. У разі false анімація буде запущена відразу ж, не встаючи в чергу.

– *specialEasing* – дозволяє встановити різні значення easing, для різних CSS-параметрів. Задається об’єктом у форматі параметр: значення.

Приклад використання властивостей анімації

```
<!DOCTYPE html>
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    var div = $("div");
    div.animate({height: '300px', opacity: '0.4'}, "slow");
    div.animate({width: '300px', opacity: '0.8'}, "slow");
    div.animate({height: '100px', opacity: '0.4'}, "slow");
    div.animate({width: '100px', opacity: '0.8'}, "slow");
  });
});
</script>
</head>
<body>

<button>Start Animation</button>
<p>By default, all HTML elements have a static position, and cannot be moved. To
manipulate the position, remember to first set the CSS position property of the
```

element to relative, fixed, or absolute!</p>
<div
***style="background:#98bf21;height:100px;width:100px;position:absolute;"> ***

</body>

Результат виконання зазначеного коду наведений на рис. 5.1.

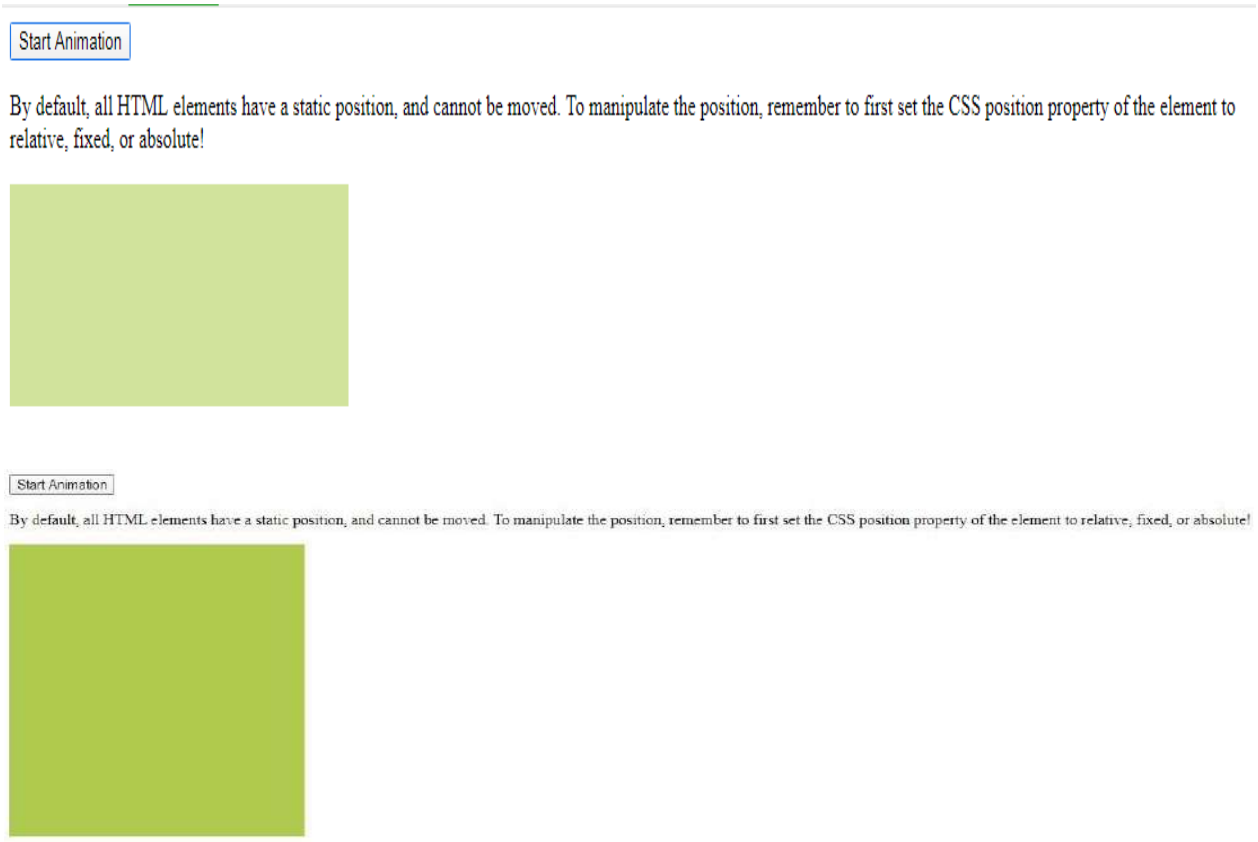


Рисунок 5.1 – Приклад стандартної анімації з використання jquery

Виконання декількох анімацій задається за допомогою властивості propertis

Задається об’єктом у форматі css-властивість: значення. Це дуже схоже на завдання групи параметрів у методі.css (), проте, properties має більш широкий діапазон типів значень. Вони можуть бути задані не тільки у вигляді звичних одиниць: чисел, пікселів, відсотків тощо, але ще і щодо : {height: "+ = 30", left: "- = 40"} (збільшіть висоту на 30 пікселів і змістіть вправо на 40). Крім того, можна задавати значення «hide», «show», «toggle», які сховають, покажуть або змінять видимість елемента на протилежну, за рахунок параметра, до якого вони застосовані. Наприклад,

$('div').animate(
{

```
   opacity: "hide",
   height: "hide"
  },
5000);
```

Приховає div-елементи, за рахунок зменшення прозорості та зменшення висоти (згортанням) елемента.

Зауваження 1: Відзначимо, що в параметрі properties можна вказувати тільки ті css-властивості, які задаються за допомогою числових значень. Наприклад, властивість background-color використовувати не слід.

Зауваження 2: Величини, які в css пишуться з використанням дефіса, мають бути вказані без нього (Не margin-left, а marginLeft).

Дія обробника завершення анімації

Функція, задана як обробник завершення анімації, не отримує параметрів, проте, всередині функції, змінна this міститиме DOM-об’єкт анімованого елемента. Якщо таких елементів декілька, то обробник буде викликаний окремо для кожного елемента.

Цей параметр визначає динаміку виконання анімації – чи буде вона проходити з уповільненням, прискоренням, рівномірно або якось іще. Параметр easing задають за допомогою функції. У стандартному jQuery доступні лише дві такі функції: 'linear' і 'swing' (для рівномірної анімації та анімації з прискоренням). За замовчуванням, easing дорівнює 'swing'.

Існує можливість задавати різні значення easing для різних css-властивостей з виконанням однієї анімації. Для цього потрібно скористатися другим варіантом функції animate () і задати опцію specialEasing. Наприклад:

```
$('#clickme').click(function() {
  $('#book').animate({
    opacity: 'toggle',
    height: 'toggle'
  }, {
    duration: 5000,
    specialEasing: {
      opacity: 'linear',
      height: 'swing'
    }
  });
});
```

У цьому випадку зміна прозорості відбуватиметься рівномірно (linear), а висота змінюватиметься з розгоном на початку і невеликим гальмуванням в кінці (swing).

З натисканням на кнопку зробимо деякі маніпуляції з елементом, використовуючи метод animate:

```
<!DOCTYPE html>
<html>
<head>
  <style>
    div {
      background-color:#bca;
      width:100px;
      border:1px solid green;
    }
  </style>
  <script src="http://code.jquery.com/jquery-latest.min.js"></script>
</head>
<body>
  <button id="go">» З'їж пиріжок </ button>
 <div id = "block"> Аліса </ div>
 <script>
 // Зробимо зміна декількох css-величин в ході однієї анімації.
$ ( "# go"). click (function () {
 $ ( "# block"). animate ({
        width: "70%", // ширина стане 70%
        opacity: 0.4, // прозорість буде 40%
        marginLeft: "0.6in", // відступ від лівого краю елемента стане рівним 6 дюймам
        fontSize: "3em", // розмір шрифту збільшиться в 3 рази
        borderWidth: "10px" // товщина рамки стане 10 пікселів
        }, 1500); // анімація відбуватиметься 1,5 секунди
        });
 </ script>
</ body>
</ html>
```

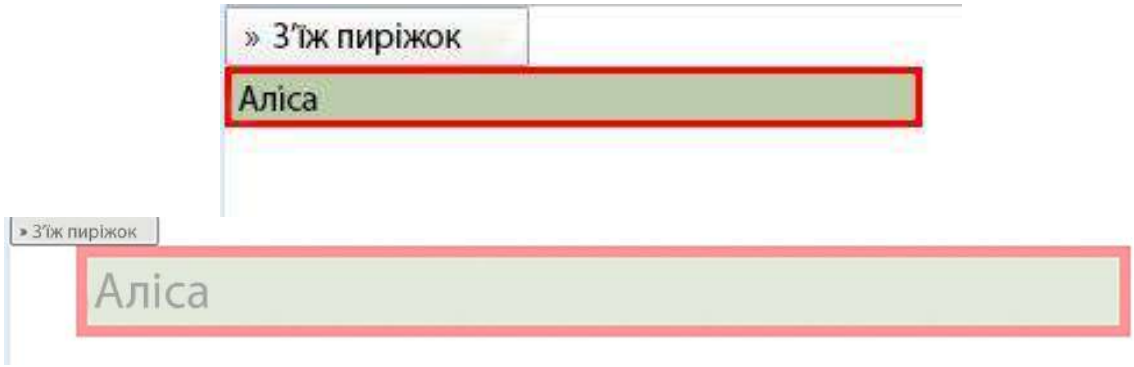


Рисунок 5.2 – Результат дії коду зі зміною прозорості

Черга майбутніх функцій

Функція .queue () повертає або змінює (залежно від переданих параметрів) чергу функцій.

Річ у тім, що до елементів сторінки jQuery прив'язує одну або декілька черг функцій. Кожна функція в такій черзі автоматично викликається, коли завершиться попередня. Функції, які виконують анімаційні ефекти, поміщаються в таку чергу автоматично. Метод queue () дозволяє отримувати і змінювати черги функцій. Є два варіанти його використання:

queue ([queueName]): jQueryv: 1.2.

Дана дія повертає чергу функцій (у вигляді масиву), що належать до обраних елементів. За допомогою необов'язкового параметра queueName можна вказати ім'я черги, яка має бути повернена. За замовчуванням це стандартна черга «fx»:

.queue ([queueName], newQueue): jQueryv: 1.2.

Вона замінює чергу з ім'ям queueName, на newQueue. За замовчуванням, буде замінена стандартна черга «fx». Метод .queue ([queueName], callback (next)):jQueryv:1.2 встановлює процедуру callback в кінець черги.

Для правильного функціонування черги в кінці методу callback необхідно виконувати одну з двох операцій: next (); або $ (this) .dequeue ().

Ці операції потрібні для запуску виконання наступних елементів черги. Розглянемо найпростіший випадок утворення черги:

– $ ('# foo'). slideUp (). fadeIn (); задасть елементу з ідентифікатором foo дві анімації в результаті. Елемент з ідентифікатором foo почне виконувати ефект slideUp (), а метод, що виконує ефект fadeIn () буде поміщений в чергу і почне своє виконання тільки після завершення попереднього методу.

Створимо чергу функцій, що виконують різні ефекти над div-елементами. Після двох з них встановимо свої функції (які змінюватимуть колір елемента), за допомогою методу queue () ;:

```
<! DOCTYPE html>
        <html>
        <head>
         <style>
         div {
        margin: 3px; width: 40px; height: 40px;
        position: absolute; left: 0px; top: 30px;
        background: green; display: none;
         }
         div.newcolor {background: blue;}
         </ style>
         <script src = "http://code.jquery.com/jquery-latest.min.js"> </ script>
        </ head>
        <body>
         Click here ...
         <div> </ div>
         <script>
         $ (document.body) .click (function () {
         $ ( "div"). show ( "slow"); // зробимо елемент видимим
        $ ( "div"). animate ({left: '+ = 200'}, 2000); // перемістимо елемент на 200 пікселів правіше
         $ ( "div"). queue (function () {// додамо нову функцію в чергу
$ (this) .addClass ( "newcolor"); // додамо клас елемента, за рахунок чого він змінить колір
        $ (this) .dequeue (); //! продовжимо чергу!
         });
$ ( "div"). animate ({left: '- = 200'}, 500); // перемістимо елемент на 200 пікселів лівіше
 $ ( "div"). queue (function (next) {// додамо нову функцію в чергу
        $ (this) .removeclass ( "newcolor"); // видалимо клас у елемента, за рахунок чого він поверне колишній колір
        next (); //! продовжимо чергу!
         });
        $ ( "div"). slideup (); // приховуємо елемент
         });
 </ script>
        </ body>
        </ html>
```

В результаті після виконання перших двох анімацій, заданих методами show () і animate () (рядки 19 і 20), буде виконана задана нами функція, яка

додасть новий клас елементів div. Потім буде виконана ще одна анімація (метод animate (), рядок 25), після чого буде викликаний ще один заданий нами метод, який видалить заданий нами клас у div-елементів.

Розглянемо почергове виконання функцій.toggle ().

По черзі виконує одну з декількох заданих дій. Має чотири варіанти використання:

.toggle (handler1 (eventObject), handler2 (eventObject), [handler3 (eventObject)]).

По черзі виконує одну з двох або більше заданих функцій handler, у відповідь на «клік» по елементу:

.toggle ([duration], [callback]).

Змінюється видимість обраних елементів на протилежну (показує/приховує).

Duration – тривалість виконання анімації. Може бути задана в мілісекундах або рядковим значенням 'fast' або 'slow' (200 і 600 мілісекунд). Callback – функція, яка буде викликана після завершення анімації, .toggle ([duration], [easing], [callback]), easing – зміна швидкості появи/зникнення (буде вона сповільнюватися до кінця виконання або, навпаки, прискориться).

Властивість .toggle (showOrHide) показує (showOrHide = true) або тільки прибирає з екрана (showOrHide = false) вибрані елементи на сторінці.

Зауваження: Зазначимо істотну відмінність першого варіанта використання методу toggle (), від двох інших: у першому варіанті метод використовується як обробник події onClick в обраних елементів, у той час як два інших варіанти викликаються безпосередньо для зміни стану обраних елементів.

```
<! DOCTYPE html>
<html>
<head>
 <style>
 ul {margin: 10px; list-style: inside circle; font-weight: bold;}
 li {cursor: pointer;}
 </ style>
 <script src = "http://code.jquery.com/jquery-latest.min.js"> </ script>
</ head>
<body>
 <ul>
 <li> Прийшов </ li>
 <li> Побачив </ li>
 <li> Переміг </ li>
 <li> Розслабився </ li>
```

```
 <li> І відхопив ... </ li>
 </ ul>
<script>
 $ ( "li"). click (function () {
 $ (this) .toggle (1000, function () {
 $ (this) .toggle (true);
 })
 });
</ script>
</ body>
</ html>
```

В результаті, з клацанням по елементу списку, він на початку почне плавно зникати (за рахунок toggle (1000, func ...), а після повного зникнення буде викликаний оброблювач закінчення анімації. Він поверне видимість зниклому елементу списку.

Почергова поява прихованих елементів здійснюється методом .fadeToggle ()

Виклик цього методу призводить до плавного зникнення (якщо елемент не прихований) або появи (якщо елемент прихований) обраних елементів на сторінці, за рахунок зміни прозорості. Зазначимо, що після приховування елемента його css-властивість display стає рівною none, а перед появою вона отримує своє колишнє значення назад. Метод має один варіант використання: .fadeToggle ([duration], [easing], [callback]).

Duration – тривалість виконання анімації (появи або приховування). Може бути задана в мілісекундах або рядковим значенням 'fast' або 'slow' (200 і 600 мілісекунд). За замовчуванням анімація буде відбуватися за 400 мілісекунд. Easing – зміна швидкості анімації (буде вона сповільнюватися до кінця виконання або, навпаки, прискориться). Callback – функція, задана як обробник завершення анімації (появи або приховування). Їй не передається жодних параметрів, проте, всередині функції змінна this міститиме DOM-об’єкт анімованого елемента. Якщо таких елементів декілька, то обробник буде викликаний окремо для кожного елемента.

Можна приховувати і показувати елементи за допомогою зміни розмірів і прозорості, за допомогою функцій show (), hide ().

Якщо ви хочете лише показати або лише приховати елементи за рахунок зміни їх прозорості, скористайтеся методами .fadeIn (), .fadeOut (). Розглянемо наступний код (рис. 5.3):

```
<! DOCTYPE html>
<html>
<head>
 <script src = "http://code.jquery.com/jquery-1.4.4.js"> </ script>
</ head>
<body>
 <button> Вуаля! </ button> <br> <br>
 <div>
 <img src = "/ images / ball4.jpg" style = "height: 150px; float: right; margin: 5px">
 <b> Правило. </ b> Ё має використовуватися: у випадках можливих різночитань; в словниках;
в книгах для тих, хто вивчає російську мову (тобто дітей та іноземців); для правильного прочитання рідкісних топонімів, назв чи прізвищ. У всіх інших випадках наявність літери Ё тільки ускладнює читання. Вона погано виглядає, зате добре звучить.
  </div>
<script>
   $("button").click(function () {
     $("div").fadeToggle("slow");
   });
</script>
</body>
</html>
```

Поява та зникнення елементів за рахунок прозорості

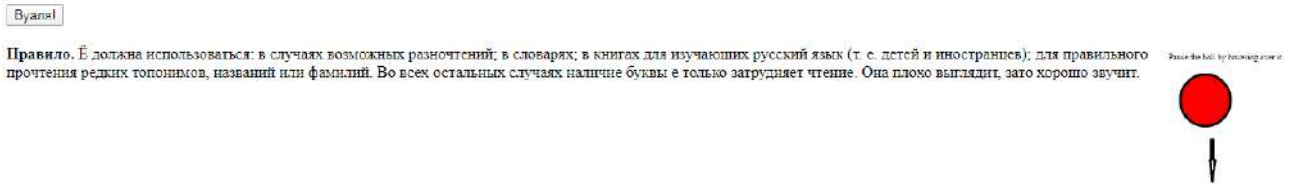


Рисунок 5.3 – Демонстрація коду, наведеного вище
(з натисканням кнопки зображення з текстом зникає)

За допомогою функцій **.fadeIn (), .fadeOut ()** можна показувати і приховувати елементи на сторінці, за рахунок плавної зміни прозорості. Зазначимо, що після приховування елемента його css-властивість display автоматично стає рівною none, а перед появою вона отримує своє колишнє значення. Методи мають два варіанти використання:

.fadeIn ([duration], [callback]) .fadeOut ([duration], [callback]): jQueryv: 1.0.

Duration – тривалість виконання анімації (появи або приховування). Може бути задана в мілісекундах або рядковим значенням 'fast' або 'slow' (200 і 600 мілісекунд). За замовчуванням анімація буде відбуватися за 400 мілісекунд. Callback – функція задана як обробник завершення анімації (появи або приховування). Їй не передається жодних параметрів, проте, всередині функції змінна this міститиме DOM-об'єкт анімованого елемента. Якщо таких елементів декілька, то обробник буде викликаний окремо для кожного елемента.

Спосіб виклику виглядає так, тобто, .fadeIn ([duration], [easing], [callback]), .fadeOut ([duration], [easing], [callback]): duration – див. вище, easing – зміна швидкості анімації (буде вона сповільнюватися до кінця виконання або, навпаки, прискориться), callback – див. вище.

Приклади використання показано нижче:

$ ("# LeftFit").FadeOut ()	елемент з ідентифікатром leftFit «розчиниться» за 400 мс.
$ ("# LeftFit").FadeIn ()	елемент з ідентифікатром leftFit «проясниться» за 400 мс.
$ ("#LeftFit").FadeOut (300)	протягом 1/3 секунди елемент з ідентифікатором leftFit зникне.
$ ("#LeftFit").FadeIn ("slow")	протягом 600 мс з'явиться елемент з ідентифікатором leftFit.

```
<! DOCTYPE html>
<html>
<head>
 <style>
 #clickme {width: 160px; border: 1px solid gray; padding: 3px; background-color: #eee; cursor: pointer}
 </ style>
 <script src = "http://code.jquery.com/jquery-latest.min.js"> </ script>
</ head>
<body>
<div id = "clickme">
 Отримайте унікальну методику збагачення, всього за 100 доларів
</ div>
<img id = "robot" src = "/ tags / images / 100usd.jpg" alt = "" />
<div id = "result"> </ div>
<script>
$ ( '# clickme'). click (function () {
 $ ( '# robot'). fadeOut ( 'slow', function () {
 $ ( '# result'). html ( "Ну, як завжди ...");
 });
```

```
});
</ script>
</ body>
</ html>
```

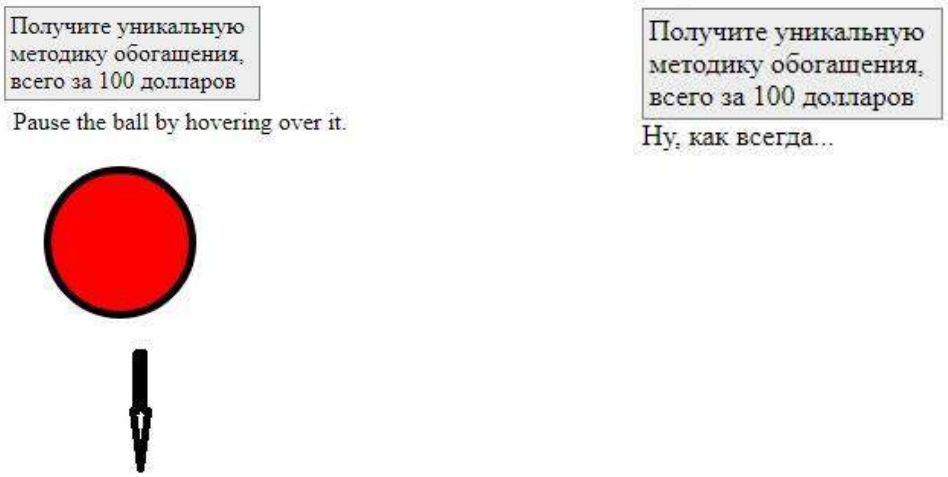


Рисунок 5.4 – Демонстрація коду, наведеного вище

Поява та приховування елементів

За допомогою функцій .show () .hide () можна плавно показувати і приховувати вибрані елементи на сторінці за рахунок зміни розміру і прозорості. Зазначимо, що після приховування елемента його css-властивість display стає рівною none, а перед появою вона отримує своє колишнє значення назад. Метод має три варіанти використання

.show (), .hide ():

– миттєво показує/приховує вибрані елементи, встановивши їх css-властивість display в none, не змінюючи при цьому їх прозорість і розміри:

.show (duration, [callback]) .hide (duration, [callback]).

Duration – тривалість виконання анімації (появи або приховування). Може бути задана в мілісекундах або рядковим значенням 'fast' або 'slow' (200 і 600 мілісекунд). Якщо цей параметр не заданий, анімація відбуватиметься миттєво, елемент просто з’явиться/зникне.

Callback – функція задана як обробник завершення анімації (появи або приховування). Їй не передається жодних параметрів, проте, всередині функції, змінна this міститиме DOM-об’єкт анімованого елемента. Якщо таких елементів декілька, то обробник буде викликаний окремо для кожного елемента.

Приклади використання методів .show ([duration], [easing], [callback]) і .hide ([duration], [easing], [callback]):

$ ("# LeftFit"). Hide ()	миттєво приховає елемент з ідентифікатором leftFit.
$ ("# LeftFit"). Show ()	миттєво покаже елемент з ідентифікатором leftFit.
$ ("# LeftFit"). Hide (300)	протягом 1/3 секунди приховає елемент з ідентифікатором leftFit.
$ ("# LeftFit"). Show ("slow")	протягом 600 мілісекунд поверне видимість елементу з ідентифікатором leftFit.

```
<! DOCTYPE html>
<html>
<head>
 <style>
 #clickme {width: 300px; border: 1px solid gray; padding: 3px; cursor: pointer}
 </ style>
 <script src = "http://code.jquery.com/jquery-latest.min.js"> </ script>
</ head>
<body>
<div id = "clickme">
 Натисни на напис, отримаєш результат ...
</ div>
<img id = "robot" src = "/ tags / images / robot.jpg" alt = "" />
<div id = "result"> </ div>

<script>
$ ( '# clickme'). click (function () {
 $ ( '# robot'). hide ( 'slow', function () {
 $ ( '# result'). html ( "... але твоя мрія не здійсниться. <br>" +
 "Натиснув на напис, але не особливо радий - <br>" +
 "З сайтом належить всю ніч возитися & nbsp; :-(");
 });
});
</ script>
</ body>
</ html>
```

Властивості $ .fx.intervalv: містить часовий проміжок (в мілісекундах) між кадрами анімації. За замовчуванням вона дорівнює 13. Властивість є глобальною для будь-якої анімації бібліотекою jQuery.

Збільшимо значення $ .fx.interval до 100 мілісекунд. В результаті анімації будуть відбуватися тільки 10 разів в секунду:

```
<! DOCTYPE html>
<html>
<head>
 <style>
 div {width: 50px; height: 30px; margin: 5px; float: left; background: green;}
 </ style>
 <script src = "http://code.jquery.com/jquery-1.4.4.js"> </ script>
</ head>
<body>
 <p> <input type = "button" value = "go" /> </ p>
 <div> </ div>
 <script>
 jQuery.fx.interval = 100;
 $ ( "input"). click (function () {
 $ ( "div"). toggle (3000);
 });
 </ script>
</ body>
</ html>
```

Скасування анімацій відбувається jQuery.fx.off :.

Встановивши цю властивість в true, ви вимкнете всі анімації, які можна виконувати за допомогою jQuery. Для того щоб анімації запрацювали знову, необхідно встановити цю властивість назад в false.

Простий приклад:

```
jQuery.fx.off = true;
$ ( '# foo'). slideUp (). fadeIn (); // виклик цих і всіх наступних анімацій ні до чого не приведе.
```

5.4. Контрольні запитання та завдання

1. Поясніть метод show ().
2. Поясніть значення параметрів методу slideDown ().
3. Поясніть значення параметрів методу slideToggle ().
4. Яке призначення методу hide ()?
5. Який метод перемикає вибраний пункт одного стану в інший, залежно від його поточного стану?
6. Який метод змінює властивість елемента і після приховує елемент?
7. Якими методами можна управляти анімацією?
8. Якими методами видаляються всі властивості анімації?
9. Як за допомогою бібліотеки JQuery можна змінювати тривалість і швидкість виконання анімаційних сюжетів?

5.5. Завдання для самостійного розв’язання

1. Завантажте бібліотеку http://code.jquery.com/jquery-1.4.4.jsабо іншу версію на комп'ютер. Створіть каталог зі скачаною бібліотекою і розташуйте її у внутрішньому каталозі щодо розміщення файлів з кодами, які відповідають рисункам 5.2-5.5. Файли рисунків, які використовуються, розташуйте в каталозі з вихідними кодами (src = "/ tags / images / 100usd.jpg").

2. Використовуючи коди, дії яких подано на рис. 5.2.–5.4, створіть файли з їх вмістом і перевірте їх виконання в браузерах Opera, Firefox, Chrome.

3. Додайте зміни в код, який відповідає рис. 5.3, щоб червоний м’яч був створений за допомогою коду CSS.

4. Додайте зміни в код, який реалізує вміст рис. 5.4, так щоб текст відображався у вигляді рядка, що біжить.

5. Використовуючи функції .slideDown () і .slideUp (), сховайте/покажіть елементи, які з’являються в результаті дії кодів, представлених у цьому розділі.

6. Додайте зміни в код, реалізація якого подана на рис.5.2, щоб анімована фігура переміщувалася по всій ширині і висоті екрану, змінювала колір і форму, а також швидкість переміщення.

7. Використовуючи функцію .slideToggle (), згорніть та розгорніть елементи, використовуючи код, наведений у прикладах (рис. 5.2 – 5.5).

8.Використовуючи обробник події hover, додайте зміни в прикладах на (рис. 5.2–5.5) для запуску анімації.

6. ВСТУП В БІБЛІОТЕКУ CANVAS

6.1. Стислий опис бібліотеки

Canvas (англ. canvas – «полотно», укр. канвас) – елемент HTML5, призначений для створення растрового двовимірного зображення за допомогою скриптів, зазвичай мовою JavaScript. Початок відліку блока знаходиться зліва зверху. Від нього і будується кожен елемент блока. Розмір простору координат не обов'язково відображає розмір фактичної інформації площини, що відображається. За замовчуванням його ширина дорівнює 300 px, а висота 150 px.

Використовується, як правило, для відтворення графіків для статей і ігрового поля в деяких браузерних іграх. Але також може використовуватися для вбудовування відео в сторінку і створення повноцінного плеєра.

Використовується у WebGL для апаратного прискорення 3D-графіки.

Компанією Google була випущена JavaScript-бібліотека explorercanvas, яка дозволяла працювати з Canvas в браузерах IE7 і IE8.

Canvas може ускладнити завдання роботам з розпізнавання реєстраційних форм. Під час використання canvas з сервера ми завантажуємо не картинку, а набір точок (або алгоритм прорисовуваня), за якими браузер прорисовує картинку (капчу).

Сьогодні canvas частіше використовується для побудови графіків, простих анімації та ігор у браузерах. Група WHATWG пропонує використовувати canvas як стандарт для створення графіків у нових поколіннях веб-додатків.

Організація Mozilla Foundation веде проект під назвою Canvas 3D, метою якого є додати низькорівневу підтримку графічних прискорювачів для відображення тривимірних зображень через HTML-елемент canvas. Поряд з цим існують бібліотеки, що реалізують роботу з тривимірними моделями, серед яких three.js.

Canvas дозволяє розмістити на полотні: картинку, відео, текст. Залити все це суцільним кольором, або обвести контури чи навіть додати градієнт; додавати тіні, схожі на властивості css3 box-shadow і text-shadow. І, нарешті, відрисовка фігур за допомогою зазначення контрольних точок. Причому можна змінювати як ширину ліній, так і пензль рисування ліній, стиль з'єднання ліній.

Зміна висоти або ширини полотна зітре весь його вміст і всі налаштування, простіше кажучи, він створиться заново:

– початок відліку (точка 0,0) знаходиться в лівому верхньому кутку (але її можна зрушувати);

– 3d-контексту немає, є окремі розробки, але вони не стандартизовані;
– колір тексту можна вказувати аналогічно CSS, втім, як і розмір шрифту.

Приклади оптимізації

У разі, якщо вам немає необхідності перерисовувати полотно, але потрібно проводити маніпуляції з ним, то ви можете «сфотографувати» все полотно і зберегти в змінну. І працювати вже з цим рисунком, не змушуючи канвас виконувати відрисовування після кожної маніпуляції.

Якщо оновлюватися має не все зображення, а тільки його частина, то можна зтерти певну зону на полотні і рисувати її заново.

Браузери можуть оптимізувати анімації, які йдуть одночасно, зменшивши число reflow і repaint до одного, що в свою чергу призведе до підвищення точності анімації. Наприклад, анімації на JavaScript, синхронізовані з CSS transitions або SVG SMIL. Якщо виконується анімація в табі, який невидимий, браузери не продовжуватимуть перерисовування, що призведе до меншого використання CPU, GPU, пам'яті і, як наслідок, знизить витрату батареї в мобільних пристроях. Для цього використовується requestAnimationFrame.

Усі поточні браузери мають фільтр розмиття зображення з його збільшенням. Його варто використовувати, якщо ви часто попіксельно обробляєте картинку. Шляхом зменшення картинки, наприклад, удвічі і подальшого апаратного збільшення її за допомогою фільтра.

Якщо гра дозволяє окремо обробляти фон і елементи гри, то має сенс зробити два полотна один над одним.

Для очищення канви кращим засобом буде використання clearRect, проте, якщо очищувати тільки необхідні ділянки, то швидкість зросте ще більше.

Недоліки технології CANVAS:

– надмірно навантажує процесор і оперативну пам'ять;
– через обмеження збирача сміття немає можливості очистити пам'ять;
– необхідно самому обробляти події з об'єктами;
– погана продуктивність за високого розширення;
– доводиться рисувати окремо кожен елемент;
– можливість створення на сторінках спеціальних «маячків».

Canvas Fingerprinting може використовуватися для відстеження користувачів у мережі.

Переваги технології CANVAS:

– на відміну від SVG набагато зручніше мати справу з великою кількістю елементів;

– має апаратне прискорення;

– можна маніпулювати кожним пікселем;

– можна застосовувати фільтри обробки.

Використання і операції з елементом можливі тільки через JavaScript. <!doctype html>

```
<html lang="ru">
  <head>
    <title>canvas</title>
    <script src="example.js"></script>
  </head>
  <body>
    <canvas id="canvas">Цей елемент не підтримується </ canvas>
  </body>
</html>
function onLoadHandler() {
  var canvas  = document.getElementById('canvas'),
      context = canvas.getContext('2d');
                                                /*
 Далі будь-які дії над полотном
  * /
}
window.onload = onLoadHandler;
```

Canvas перекладається як полотно, тег canvas. Це елемент HTML 5, що призначений для створення растрових зображень за допомогою JavaScript. Тому, як створити це растрове зображення і йтиметься далі. Перш ніж починати пробувати свої сили в цій нелегкій справі, рекомендується вже мати базові знання про те, що таке HTML, і орієнтуватися в коді мови JavaScript.

Наведемо опис тега canvas, що входить в html, починаючи з версії 5 [6].

```
! DOCTYPE html>
<html>
 <head>
 <title>canvas</ title>
 <meta charset = "utf-8">
 <script>
 window.onload = function () {
 var drawingCanvas = document.getElementById ( 'smile');
```

```
if (drawingCanvas && drawingCanvas.getContext) {
var context = drawingCanvas.getContext ( '2d');
// Малюємо коло
context.strokeStyle = "# 000";
context.fillStyle = "# fc0";
context.beginPath ();
context.arc (100,100,50,0, Math.PI * 2, true);
context.closePath ();
context.stroke ();
context.fill ();
// Малюємо ліве око
context.fillStyle = "#fff";
context.beginPath ();
context.arc (84,90,8,0, Math.PI * 2, true);
context.closePath ();
context.stroke ();
context.fill ();
// Малюємо праве око
context.beginPath ();
context.arc (116,90,8,0, Math.PI * 2, true);
context.closePath ();
context.stroke ();
context.fill ();
// Малюємо рот
context.beginPath ();
context.moveTo (70,115);
context.quadraticCurveTo (100,130,130,115);
context.quadraticCurveTo (100,150,70,115);
context.closePath ();
context.stroke ();
context.fill ();
}
}
</ script>
</ head>
<body>
<canvas id = "smile" width = "200" height = "200">
<p>Ваш браузер не підтримує малювання.</ p>
</ canvas>
</ body>
</ html>
```

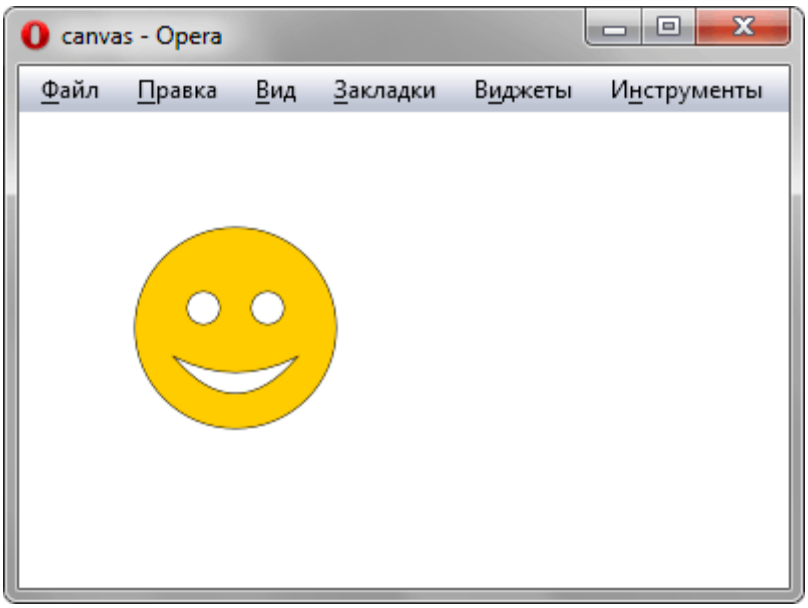


Рисунок 6.1 – Результат виконання вищевказаних дій програми

Цьому тегу притаманні стандартні атрибути і події (табл. 6.1).

Таблиця 6.1 – Опис основних атрибутів і подій тега **CANVAS**

Атрибути та події тега	**Опис**
onblur	Зображення не сфокусоване.
onchange	Зміна значення елемента форми.
onclick	Натискання лівою кнопкою мишки на елементі.
ondblclick	Подвійне натискання лівою кнопкою мишки на елементі.
onfocus	Отримання фокусу.
onkeydown	Клавіша натиснута, але не відпущена.
onkeypress	Клавіша натиснута і відпущена.
onkeyup	Клавіша відпущена.
onload	Документ завантажений.
onmousedown	Натиснута ліва кнопка мишки.
onmousemove	Переміщення курсора мишки.
onmouseout	Курсор залишає елемент.
onmouseover	Курсор наводиться на елемент.
onmouseup	Ліва кнопка мишки відпущена.
onreset	Форма очищена.
onselect	Виділено текст в полі форми.
onsubmit	Форма відправлена.
onunload	Закриття вікна.

Універсальні атрибути застосовуються практично до всіх тегів, тому виділені в окрему групу, щоб не повторювати їх для всіх тегів (табл. 6.2).

Таблиця 6.2 – Опис універсальних атрибутів

Атрибут	Опис
accesskey	Дозволяє отримати доступ до елемента за допомогою заданого поєднання клавіш.
class	Визначає ім'я класу, яке дозволяє зв'язати тег зі стильовим оформленням.
contenteditable	Повідомляє, що елемент доступний для редагування користувачем.
contextmenu	Встановлює контекстне меню для елемента.
dir	Задає напрямок і відображення тексту – зліва направо або справа наліво.
hidden	Приховує вміст елемента від перегляду.
id	Вказує ім'я стильового ідентифікатора.
lang	Браузер використовує значення параметра для правильного відображення деяких національних символів.
spellcheck	Вказує браузеру, перевіряти правопис і граматику в тексті, чи ні.
style	Застосовується для визначення стилю елемента за допомогою правил CSS.
tabindex	Встановлює порядок включення фокуса між елементами за допомогою вкладки клавіш.
title	Описує вміст елемента у вигляді підказки.

Розглянемо приклад рисування із запам'ятовуванням і повторним відтворенням зображення.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>Document</title>
</head>
<body style="margin: 0;">
    <canvas id="canvas" style="display: block;"> youre brouser</canvas>
<script>
    var
    canv = document.getElementById('canvas'),
    ctx  = canv.getContext('2d');
    // code 4
```

```
        ifMouseDown = False,
//
coords = [];
// finish code 4
        canv.width = Window.innerWidth;
        canv.height = Window.innerHeight;
/ *

// code 4

        * /
        // code 3

canv.addEventListener ( 'mousedown', function () {
isMouseDown = true;
});
canv.addEventListener ( 'mouseup', function () {
isMouseDown = false;
ctx.beginPath (); // початок шляху
// запам'ятовування відпустки мишки
coords.push ( 'mouseup');

});
//


canv.addEventListener ( 'mousemove', function (e) {
if (isMouseDown)
{
// code 4b // додавання координат в масив
        coords.push ([e.clientX, e.clientY]);
// Додає нову точку і створює лінію
ctx.lineTo (e.clientX, e.clientY);
// малює шлях
ctx.stroke ();
ctx.beginPath ();
// Створює дугу
ctx.arc (e.clientX, e.clientY, 10,0, 3.14 / 2);
ctx.fill (); // Заповнює поточне креслення (шлях)

ctx.beginPath ();
ctx.lineTo (e.clientX, e.clientY);

}
```

```
});
// code 4a
//
function clear () {
// очищення білим кольором
        ctx.fillStyle = 'White';
        ctx.fillRect (0,0, canv.width, canv.height);

        ctx.beginPath ();
        ctx.fillStyle = 'black';
        ctx.fillStyle = 'red'
}

function replay ()

// відрисовування після успішної реєстрації зображення
{
var timer = setInterval (function () {
// якщо щось нарисовано
if (! coords.length)
{
clearInterval (timer); sr
ctx.beginPath ();
return;
}
//
var crd = coords.shift (),
e = {clientX: crd [ "0"], clientY: crd [ "1"]}
// code 4b
ctx.lineTo (e.clientX, e.clientY);
ctx.stroke ();

ctx.beginPath ();
var er = 10;
var er1 = 0;
ctx.arc (e.clientX, e.clientY, er, er1, 3.14 / 2);
ctx.fill ();

ctx.beginPath ();
ctx.lineTo (e.clientX, e.clientY);

}, 10);
// 10 - час рисування
```

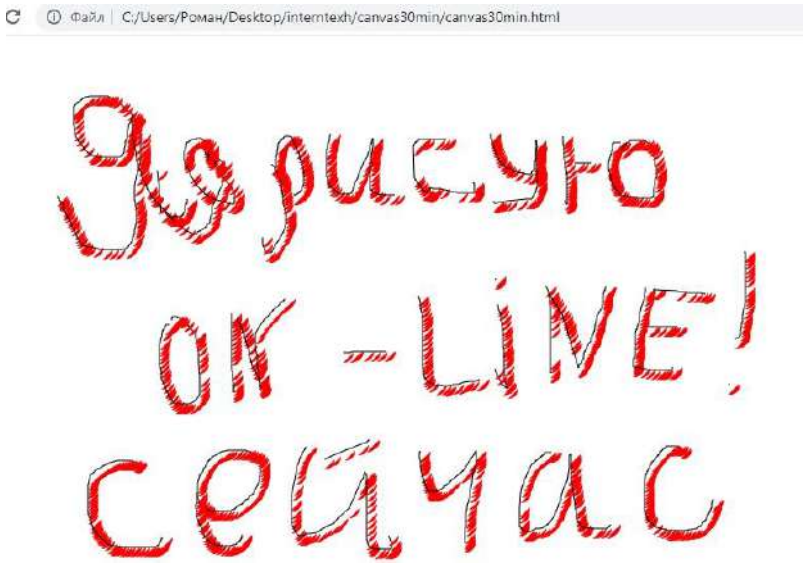
```
}

function save ()
// Якщо методу setItem () інтерфейсу Storage передати ключ і значення, то в сховище // буде додано відповідне ключу значення, або, якщо запис вже є в // сховище, то значення за ключем буде оновлено
// Інтерфейс Storage з Web Storage API надає доступ для session storage або // local storage для конкретного домену, дозволяючи вам, наприклад, додавати, змінювати // або видаляти збережені елементи даних.
// JSON.stringify
// let user = {name: "John", age: 25, roles: {isAdmin: false, isEditor: true}};
//alert(JSON.stringify(user, null, 2)); / * Відступ в 2 пробілу:
// { "name": "John", "age": 25, "roles": { "isAdmin": false, "isEditor": true
//}}

// JSON (JavaScript Object Notation) - це загальний формат для подання значень і
// об'єктів. Його опис задокументовано в стандарті RFC 4627. Спочатку він був // створений для JavaScript, але багато інших мов також мають бібліотеки, які можуть // працювати з ним. Таким чином, JSON легко використовувати для обміну даними, коли // клієнт використовує JavaScript, а сервер написаний на Ruby / PHP / Java або будь-якою іншою // мовою.
// JavaScript надає методи:
//JSON.stringify метод для перетворення об'єктів в JSON.
//JSON.parse метод для перетворення JSON назад в об'єкт.
// let user = '{ "name": "John", "age": 35, "isAdmin": false, "friends": [0,1,2,3]}';
// user = JSON.parse (user);
// alert (user.friends [1]); // 1

{
localStorage.setItem ( 'coords', JSON.stringify (coords));
}
// code 4 обробник події натискання на кнопку 83 -код клавіші -s
// 82-код клавіші -r // 67 - код клавіші з

document.addEventListener ( 'keydown', function (e) {
        // натиснули букву s
if (e.keyCode == 83)
        {

                // save
                save ();
```

```
            console.log ( 'Saved')

        }
// натиснули букву r
 if (e.keyCode == 82)
 {
        // replay
 console.log ( 'Replaying ...');
 coords = JSON.parse (localStorage.getItem ( 'coords'));
 clear ();
 replay ();
 }

 if (e.keyCode == 67)
        // cleare натиснули букву c
 {

 clear ();
 console.log ( 'Cleared');
 }
 });
</ script>
</ body>
</ html>
```

Рисунок 6.2 – Приклад рисування з використанням тега canvas

Задає початковий сюжет

Повторне рисування після збереження



Дорисовування, відтворення фрагмента після збереження

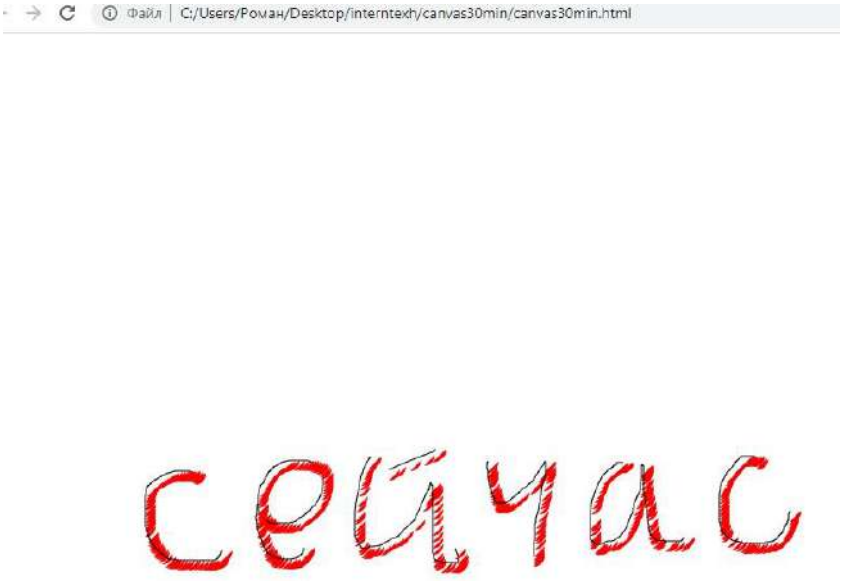


Рисунок 6.3 – Результат виконання програми рисування зі збереженням зображення

Таблиця 6.3 – Опис методів **CANVAS**

Метод	Опис
fill ()	Заповнює поточне креслення (шлях)
stroke ()	Насправді рисує шлях, який ви визначили
beginPath ()	Починає шлях або скидає поточний шлях
moveTo ()	Переміщує шлях до зазначеної точки на полотні, не створюючи лінію
closePath ()	Створює шлях від поточної точки до початкової точки
lineTo ()	Додає нову точку і створює лінію до цієї точки від останньої зазначеної точки на полотні
clip ()	Обрізає область будь-якої форми і розміру з вихідного полотна
quadraticCurveTo ()	Створює квадратичную криву Безьє
bezierCurveTo ()	Створює кубічну криву Безьє
arc ()	Створює дугу / криву (використовується для створення кіл або частин кіл)
arcTo ()	Створює дугу / криву між двома дотичними
isPointInPath ()	Повертає true, якщо зазначена точка знаходиться в поточному шляху, інакше false
stroke ()	Метод stroke () настравді рисує фігуру, контур якої було поставлено раніше. За замовчуванням фігура буде нарисована чорним кольором.

Методи трансформації

Таблиця 6.4 – Опис методів трансформації **CANVAS**

Метод	Опис
fill ()	Заповнює поточне креслення (шлях)
scale ()	Масштабує поточне креслення
translate ()	Перевизначає (0,0) позицію на полотні
transform ()	Замінює поточну матрицю перетворення для креслення, потім запускається
setTransform ()	Скидає поточне перетворення в одиничну матрицю.
scale ()	Масштабує поточне креслення більше або менше
rotate ()	Повертає поточне креслення

Таблиця 6.5 – Опис методів **CANVAS** для текстів

Метод	Опис
textAlign	Встановлює або повертає поточне вирівнювання для текстового вмісту
textBaseline	Встановлює або повертає поточну текстову базову лінію, яка використовується під час рисування тексту
fillText ()	Рисує «заповнений» текст на полотні
strokeText ()	Рисує текст на полотні (без заливки)
measureText ()	Заміряє розмір тексту

Таблиця 6.6 – Опис методів **CANVAS** для рисування зображень

Метод	Опис
drawImage ()	Рисування зображення Рисує зображення, полотно або відео на полотні
createImageData ()	Створює новий порожній об’єкт ImageData
getImageData	Повертає об’єкт ImageData, який копіює дані пікселів для зазначеного прямокутника на полотні
putImageData ()	Поміщує дані зображення (із зазначеного об’єкта ImageData) назад на полотно
globalAlpha	Встановлює або повертає поточне значення альфа або прозорості креслення
globalCompositeOperation	Встановлює або повертає спосіб рисування нового зображення на існуючому зображенні
save ()	Зберігає стан поточного контексту
restore ()	Повертає раніше збережений стан колії та атрибути
getContext ()	Створити подію ()
toDataURL ()	Отримати інтернет-адресу

6.2. Контрольні запитання та завдання

1. Методи бібліотеки для роботи з текстами.
2. Методи бібліотеки для трансформації.
3. У чому відмінність методів CSS I jquery?
4. Призначення тега CANVAS.
5. Особливості атрибутів тега CANVAS.

6.3. Завдання для самостійного розв’язання

1. Використовуючи код, відповідний результатам, наведеним на рис. 6.1, внести зміни, що дозволять змінити швидкість рисування, ширину пензля, колір пензля і колір фону.

7. ДЕМОНСТРАЦІЯ ВИКОРИСТАННЯ АЛГОРИТМУ ТРІАНГУЛЯЦІЇ ДЕЛОНЕ В ПРОФЕСІЙНІЙ АНІМАЦІЙНІЙ ПЛАТФОРМІ GSAP

7.1. Стислий опис бібліотеки

Професійна бібліотека GSAP – це анімаційна платформа, створена свого часу для Action Script, зараз імпортована і на JS. Вміє анімувати властивості абсолютно будь-яких об’єктів, зручна у використанні.

Посилання на скачування знаходиться тут

https://cdnjs.cloudflare.com/ajax/libs/gsap/3.5.1/gsap.min.js

GSAP дозволяє легко створювати рознесену анімацію на декількох об’єктах. Анімації можуть перекриватися, запускатися в прямій послідовності або мати проміжки між часом їх запуску.

Бібліотека TweenMax застаріла в GSAP 3 на користь спрощеного об’єкта gsap. Він має більше 50 нових функцій і майже на половину розміру! </ Strong> GSAP 3 сумісний з переважною більшістю функцій GSAP 2, включаючи TweenMax.

TweenMax дозволяє буквально анімувати будь-яку властивість будь-якого об’єкта, до якого може доторкнутися JavaScript (CSS, SVG, React, Vue, Three.js, canvas, траєкторії руху, загальні об’єкти тощо). До випуску GSAP 3 TweenMax був найбільш функціональним (і популярним) інструментом анімації в арсеналі GSAP. Проте в GSAP 3 вам більше не потрібно навіть посилатися на TweenMax в своєму коді (хоча ви можете це зробити, оскільки GSAP 3 дотримується переважна більшість успадкованого коду).

Для зручності і ефективності завантаження в TweenMax 2 і більш ранніх версій включені TweenLite, TimelineLite, TimelineMax, CSSPlugin, AttrPlugin, RoundPropsPlugin, BezierPlugin і EasePack (все в одному файлі). Кожен біт цієї функціональності знаходиться в ядрі GSAP 3, що майже вдвічі менше.

Старі методи розбивки TweenMax (версії 1 і 2): TweenMax.staggerTo (), TweenMax.staggerFrom () і TweenMax.staggerFromTo () навіть не потрібні в GSAP 3, оскільки спеціальна властивість «stagger» може бути додана в будь-який код.

Основні методи даної бібліотеки докладно описані в

https://greensock.com/get-started/:	*delay (), delayedCall (), duration (), eventCallback (), from (), fromTo (), getTweensOf (), invalidate (), isActive (), kill (), killDelayedCallsTo (), killTweensOf (), pause (), paused (), play (), progress (), restart (), resume (), reverse (), reversed (), seek (), set (), startTime (), time (), timeScale (), totalDuration ( ), totalProgress (), totalTime ()*

Методи ексклюзивні для бібліотеки TweenMax

https://codepen.io/chrisgannon/pen/WNNJqXz:	*getAllTweens (), isTweening (), killAll (), killChildTweensOf (), pauseAll (), repeat (), repeatDelay (), resumeAll (), staggerFrom (), staggerFromTo (), staggerTo (), updateTo ().*

7.2. Бібліотека TweenMax.js

Перш за все завантажте бібліотеку з http://greensock.com/. Ви можете завантажити у вигляді zip-файла з проекту GIT або використовувати через cdn (мережа доставки контенту). Розпакуйте zip-пакет і відкрийте папку з бібліотекою. Усередині ви знайдете html-файл з інструкцією і html-файл документації. Потім можете перейти до мінімізованої або незтиснутої версії бібліотеки. Усередині цих папок можете легко знайти файли бібліотек для TweenMax, TweenLite, TimelineMax і TimelineLite. Також можете помітити, що є папки з іменами easing, plugins і utils. Щоб максимально використовувати цю бібліотеку, скопіюйте TweenMax.min.js і TimelineMax.min.js, папки easing, plugins і utils в папку javascript в проекті Tizen IDE, не змінюючи структуру каталогів. Наступним кроком буде додавання бібліотек у ваш проект на JavaScript.

Приклад включення

<script src = "js / TweenMax.min.js"> </ script>,
<Script src = "js / TimelineMax.min.js"> </ script>.

Зробивши це, ми можемо почати використовувати можливості програмної анімації з TweenMax.

В html-файлі прикладу додатки ми можемо знайти наступний елемент , який будемо анімувати:

*** Приклад TweenMax і TimelineMax </br> для Tizen </ div>***

У вашому файлі main.js вам потрібно вказати ідентифікатор за допомогою змінної JavaScript:

var mainText;
mainText = document.getElementById («основний текст»).

Будемо анімувати елемент в JavaScript за допомогою TweenMax. Зверніть увагу, що наступний код відрізняється від коду в нашому прикладі програми. Причина в тому, що наведений нижче код менш складний і простіший для розуміння.

TweenMax.to (mainText, 0.75, {затримка: 1, css: {color: "# ffffff"}, easy: Sine.easeOut});

Можна побачити певні логічні частини коду, що відповідають за всю анімацію. Спочатку потрібно викликати об'єкт TweenMax. Потім – вибрати функцію to () з It, тому що ми хочемо змінити поточний стан об'єкта , що містить текст, на інший стан. Потім в функції to () потрібно вказати параметри анімації.

Перший параметр – це об'єкт елемента DOM або масив об'єктів елемента DOM, які ми хочемо анімувати водночас. У прикладі ми поміщаємо сюди нашу змінну mainText, яка містить посилання на наш . Другий параметр – це тривалість нашої анімації в секундах. Це може бути значення з плаваючою комою, як у прикладі вище. Отже, тепер ми знаємо, який об'єкт ми хочемо анімувати і як довго має тривати анімація. Наступним параметром нашої функції to () має бути об'єкт, що містить деталі (пари ключ-значення) анімації. Давайте подивимося на наш приклад вище. Ми можемо ввести параметр затримки, який не є обов'язковим, але ви можете використовувати його, щоб відкласти запуск анімації на задану кількість секунд. Наступний параметр у нашому прикладі – це об'кт, що містить параметри CSS, призначені для анімації. У нашому прикладі ми змінюємо параметр кольору CSS нашого на #ffffff (білий). Крім того, ви можете вказати додаткові параметри у вигляді пар ключ-значення, розділених двокрапкою. І це не обов'язково мають бути тільки параметри CSS. Можна анімувати будь-які параметри об'єкта. Припустимо, у вашому коді є простий об'єкт JavaScript, як показано нижче.

Можна змінювати будь-яке з цих значень за допомогою TweenMax. У наступному прикладі значення змінних a, b і c зміняться за 2 секунди відповідно на 20, -40 і 12000 з використанням функції інтерполяції Sine.easeOut

```
var exampleObject = новий об'єкт ();
 exampleObject.a = 4;
 exampleObject.b = 0;
 exampleObject.c = 8;
TweenMax.to (exampleObject, 2, {a: 20, b: -40, c: 12000}, easy: Sine.easeOut);
```

Останній параметр у нашому прикладі – easy. Easy – дуже цікавий параметр, оскільки він повідомляє движку анімації TweenMax, як інтерполювати зміну цього значення протягом заданого часу. Є безліч визначених функцій на вибір. Також ви можете створювати власні функції плавності. У нашому прикладі ми вибрали інтерполяцію Sine.easeOut.

Приклад виклику

```
var myTween = new TweenMax (exampleObject, 2, {a: 20, b: -40, c: 12000},
easy: Sine.easeOut).
```

Також слід згадати, що у TweenMax є зворотні виклики. Наприклад, він може повідомляти вам про кожне оновлення анімації руху або про завершення анімації руху. Ви можете перевірити стан проміжних змінних, призначивши функцію змінної onUpdate, і перевірити, чи завершилася анімація руху, призначивши функцію змінної onComplete. У наведеному нижче прикладі показано, як обидва зворотних виклики можуть застосовуватися до анімації руху. Приклад коду:

```
var exampleObject = новий об'єкт ();
   exampleObject.a = 4;
   exampleObject.b = 0;
   exampleObject.c = 8;
var myTween = new TweenMax (exampleObject, 2, {a: 20, b: -40, c: 12000,
onUpdate: tweenUpdate, onComplete: tweenComplete}, easy: Sine.easeOut);
function tweenUpdate () {
console.log ( «a =« + exampleObject.a + », b =« + exampleObject.b + », c =« +
exampleObject.c);
}
function tweenComplete () {
console.log ( «Ура! Анімація завершена!»);
}
```

Крім використання TweenMax в анімації у веб-додатках HTML5 ви також можете використовувати бібліотеку в будь-якому ігровому середовищі або навіть в проектах WebGL.

7.3. Основи алгоритму тріангуляції Делоне

Вважають, що на множині точок на площині задана тріангуляція, якщо деякі пари точок з'єднані ребром, будь-яка кінцева грань в отриманому графі утворює трикутник, ребра не перетинаються, і граф максимальний за кількістю ребер.

Тріангуляцією Делоне називається така тріангуляція, в якій для будь-якого трикутника вірно, що всередині описаного навколо нього кола не знаходиться точок з вихідної множини.

Триангуляція Делоне – тріангуляція для заданої множини точок S на площині, за якої для будь-якого трикутника всі точки з S за винятком точок, які є його вершинами, лежать поза окружністю, описаною навколо трикутника. Позначається DT (S). Вперше описана в 1934 році радянським математиком Борисом Делоне.

Зауваження: Для заданої множини точок, в якій жодні 4 точки не перебувають на одному колі, існує саме одна тріангуляція Делоне.

Розглянемо умову Делоне. Нехай на множині точок задана тріангуляція. Вважатимемо, що деяка підмножина точок задовольняє умові Делоне, якщо тріангуляція, обмежена на цю підмножину, є тріангуляцією Делоне для неї.

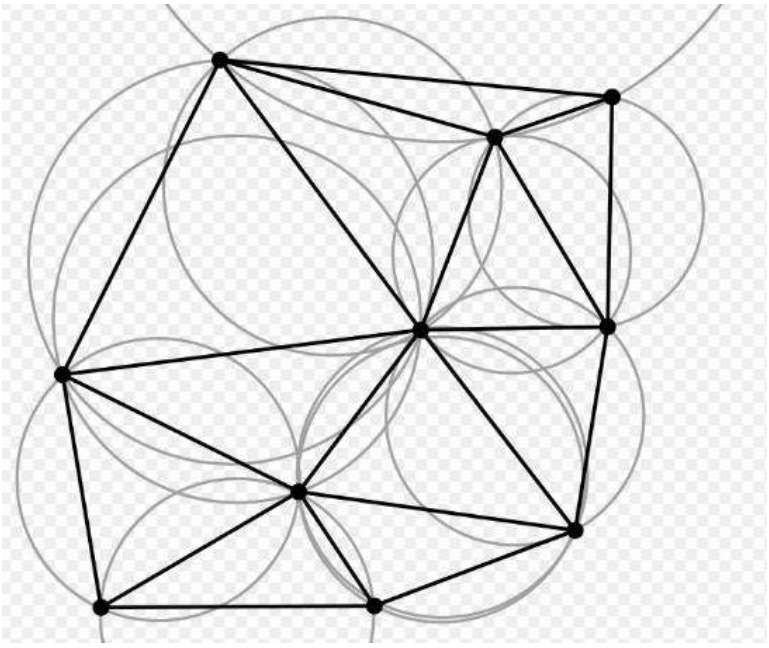


Рисунок 7.1 – Геометрична інтерпретація тріангуляції Делоне

Розглянемо критерій для тріангуляції Делоне. Виконання умови Делоне для всіх точок, що утворюють чотирикутник в тріангуляції, еквівалентне тому, що дана тріангуляція є тріангуляцією Делоне.

Зауваження: Для неопуклих чотирикутників умова Делоне завжди виконана, а для опуклих чотирикутників (вершини яких не лежать на одному колі) існує саме 2 можливі тріангуляції (одна з яких є тріангуляцією Делоне).

Завдання полягає в тому, щоб для заданої множини точок побудувати тріангуляцію Делоне.

Розглянемо опис зазначеного алгоритму.

В описі алгоритму використовуються видимі точки і видимі ребра.

Нехай задана мінімальна опукла оболонка кінцевої множини точок (ребра, що з'єднують деякі з точок так, щоб вони утворювали багатокутник, який містить всі точки множини) і точка А, що лежить поза оболонкою. Тоді точка площини називається видимою для точки А, якщо відрізок, який з'єднує її з точкою А, не перетинає мінімальну опуклу оболонку. Ребро мінімальної опуклої оболонки називається видимим для точки А, якщо його кінці видимі для А. На рис. 7.3 червоним позначені ребра, видимі для червоної точки.

Зауваження 1: Контуром тріангуляції Делоне є мінімальна опукла оболонка для точок, на яких вона побудована.

Зауваження 2: В алгоритмі видимі для доданої точки А ребра утворюють ланцюжок, тобто кілька ребер мінімальної опуклої оболонки.

Зберігання тріангуляції в пам'яті

Є деякі стандартні засоби, непогано описані в книзі Скворцова. Зважаючи на специфіку алгоритму, запропоную свій варіант. Оскільки хочеться перевіряти чотирикутник на умові Делоне, то розглянемо його будову. Кожен чотирикутник у тріангуляції – це два трикутники, які мають загальне ребро. У кожного ребра є саме два трикутники, прилеглих до нього. Таким чином, кожен чотирикутник в тріангуляції породжується ребром і двома вершинами, що знаходяться навпроти ребра в прилеглих трикутниках. Оскільки по ребру і двом вершинам відновлюються два трикутники і їх суміжність, то з усіх вищевказаних структур ми зможемо відновити тріангуляцію. Відповідно пропонується зберігати ребро з двома вершинами в множині і виконувати пошук по ребру (впорядкованої пари вершин). Ідея прямої, що замикає:

1. Відсортуємо всі точки уздовж деякої прямої.
2. Побудуємо трикутник на перших трьох точках.

3. Далі для кожної наступної точки виконуватимемо кроки, зберігаючи інваріант, що є тріангуляція Делоне для вже доданих точок і, відповідно, мінімальна опукла оболонка для них.

4. Додамо трикутники, утворені видимими ребрами і самою точкою (тобто додамо ребра з даної точки в усі кінці видимих ребер).

5. Перевіримо на умову Делоне всі чотирикутники, породжені видимими ребрами. Якщо десь умова не виконалася, то перебудуємо тріангуляцію в чотирикутнику (пам'ятаємо, що їх усього дві) і рекурсивно запустимо перевірку для чотирикутників, породжених ребрами поточного чотирикутника (адже тільки в них після зміни умова Делоне могла порушитися).

7.3.1. Стріпіфікація (поділ на смуги) полігональних моделей

На сьогодні для опису тривимірних моделей графічних об'єктів, а точніше їх поверхонь, найбільш часто використовуються тріангуляційні моделі – меші (англ. mesh – осередок мережі). Це пов'язано, насамперед, з тим, що в сучасному програмно-апаратному забезпеченні (відкритих і популярних бібліотеках тривимірної графіки, в т.ч. OpenGL і DirectX) основним графічним примітивом є трикутник.

В даному пункті вирішується дещо спільне завдання, ніж у попередніх розділах. Тут розглядається задача стріпіфікації не мешів (триангуляційних моделей), а довільних полігональних моделей – моделей поверхонь, що складаються з наборів суміжних багатокутників. Для таких полігональних моделей завдання стріпіфікації полягає у тріангуляції кожного нетрикутного багатокутника і генерації смуг трикутників.

Оскільки більшість багатокутників можна тріангувати різними способами, то завдання стріпіфікації полігональних моделей має дещо більшу свободу у виборі смуг трикутників, а тому і дозволяє отримати кращий результат, ніж під час стріпіфікації тільки тріангуляційних моделей.

Для стріпіфікації полігональних моделей часто використовуються такі основні підходи:

1. Статична тріангуляція. Тріангуляція всіх нетрикутних багатокутників і подальша стріпіфікація загального мешу будь-яким алгоритмом, з наведених у попередніх розділах.

2. Повна динамічна тріангуляція зустрічних полігонів. Під час генерації смуг SGI-алгоритмом (або його модифікаціями), якщо чергова смуга виходить не в трикутний полігон, то цей полігон тріангулюється, причому способом, який дозволяє включити в смугу максимальну кількість або навіть всі

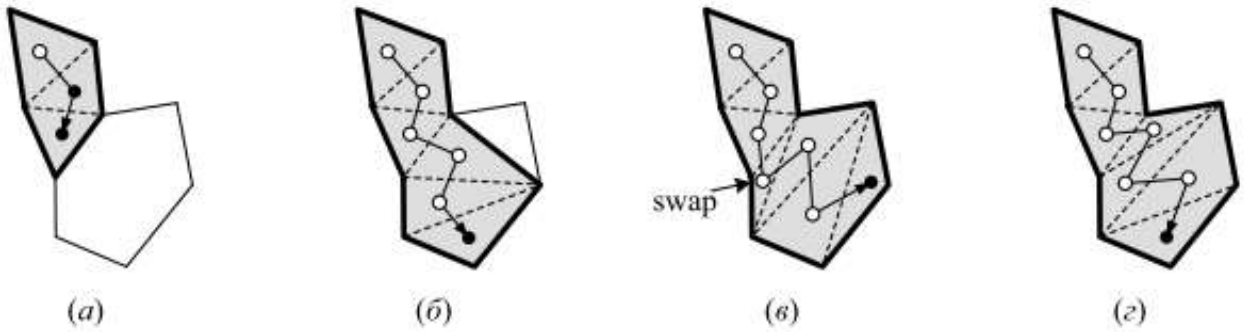
трикутники полігону і за можливості не потребує команд swap (рис. 7.3, а–в). Якщо полігон є опуклим, то в смугу можна включити всі його трикутники, а команди swap не потрібні (рис. 7.3, г).

Цей (динамічний) підхід дає результат в середньому на 15% кращий, ніж попередній (статичний) [7].

3. Часткова динамічна тріангуляція зустрічних полігонів. Відмінність даного підходу від попереднього в тому, що зі входом смуги в черговий полігон може виконуватися не повна його тріангуляція, а залежно від ситуації, тільки деяка його частина. Коли смуга входить у полігон, на основі ваг слід визначити найбільш переважну сторону полігону, через яку смуга має вийти. І тільки після цього слід побудувати мінімальну (за кількістю трикутників) смугу між вхідною та вихідною сторонами полігону.

Даний підхід зазвичай дає результат гірший, ніж попередній, і лише в рідкісних випадках – кращий.

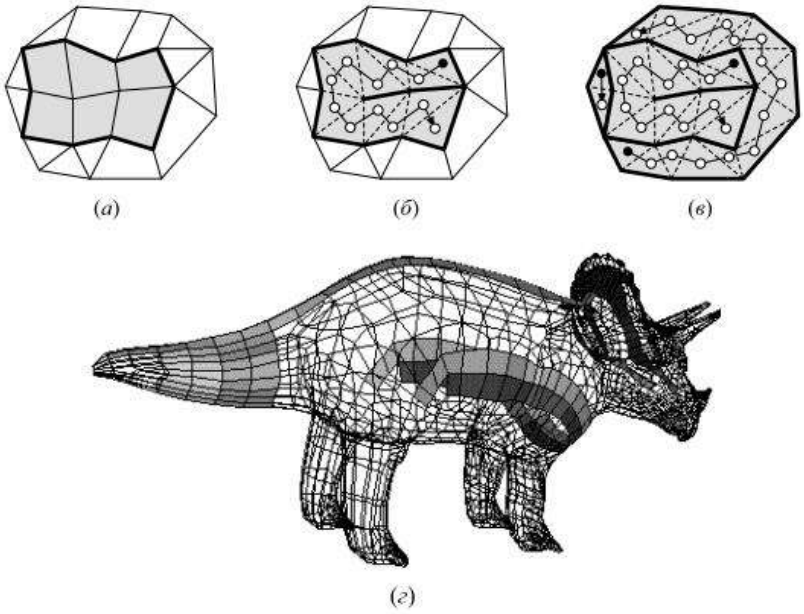
4. Стріпіфікація чотирикутних областей чотирикутників (Алгоритм STRIPE). Даний підхід застосуємо для полігональних моделей поверхонь, які складаються переважно з чотирикутників і частково трикутників, що часто зустрічаються на практиці.



(а) – смуга підійшла до багатокутника; *(б)* – невдала тріангуляція з неповним включенням в смугу; *(в)* – тріангуляція, що потребує використання команди swap; *(г)* – тріангуляція, що не потребує використання команди swap

Рисунок 7.3 – Варіанти тріангуляції полігонів із виходом смуги в полігон

Суть підходу в тому, що спочатку необхідно виділити максимально великі патчі – чотирикутні області суміжних чотирикутників (рис. 7.4, а), при цьому область має містити не менше п’яти чотирикутників. Потім кожну таку область необхідно розбити на одну смугу (рис. 7.4, б). Решту ділянок необхідно тріангулювати будь-яким попереднім способом, заснованим на SGI-алгоритмі (рис. 7.4, в). На рис. 7.4, г наведено приклад виділення чотирикутних областей чотирикутників для подальшої стріпіфікації.



(а) – виділення патчів – чотирикутних областей чотирикутників;
(б) – стріпіфікація патчів; *(в)* – стріпіфікація інших полігонів;
(г) – приклад стріпіфікациі полігональної моделі трицератопса

Рисунок 7.4 – Стріпіфікація чотирикутних областей чотирикутників

7.4. Приклад використання бібліотеки canvas з бібліотекою TweenMax.js

Розглянемо приклад з ілюстрацією. Рисунок 7.5 показує, як завантажене зображення «руйнується» з натисканням мишки в довільній точці. Код, який дозволяє реалізувати даний ефект в браузері, наведено в таблиці нижче.

Рисунок 7.5 – Приклад використання бібліотеки canvas, GSAP
і методу тріангуляції Делоне для створення ефекту «руйнування зображення»,
вихідне зображення

Рисунок 7.6 – Приклад використання бібліотеки canvas, GSAP
і методу тріангуляції Делоне для створення ефекту «руйнування зображення»
натисканням мишки на центрі зображення

Текст програми реалізує ефект, наведений на рисунках 7.5–7.6 ***<!DOCTYPE html>*** ***<html>*** ***<head>*** ***<title></title>*** ***<meta charset="utf-8">*** ***<meta name="viewport" content="width=device-width, initial-scale=1">*** ***<link type="text/css" rel="stylesheet" href="delaunay.css">*** ***<script src= "delaunay.js" ></script>*** ***</head>*** ***<body>*** *** *** ***<script type="text/javascript" src= "TweenMax.min.js" ></script>*** ***</body>*** ***</html>***	Фрагмент розмітки, з підключеними файлами стилей delaunay.css. Фірмової бібліотеки TweenMax.min.js і файла реалізує метод тріангуляції Делоне – "delaunay.js"
<script type="text/javascript"> ***const TWO_PI = Math.PI * 2;*** ***var images = [],*** ***imageIndex = 0;*** ***var image,*** ***imageWidth = 768,*** ***imageHeight = 485;*** ***var vertices = [],***	

```
    indices = [],
    fragments = [];

var container = document.getElementById('container');
var clickPosition = [imageWidth * 0.5, imageHeight * 0.5];

// Підключення бібліотеки TweenMax
window.onload = function() {
    TweenMax.set(container, {perspective:500});

    // images from reddit/r/wallpapers

    var urls = [ "crayon.jpg", "spaceship.jpg", "dj.jpg",
"chicken.jpg" ],
    /*
*/
        /*
 var urls = [
            'https://s3-us-west-
2.amazonaws.com/s.cdpn.io/175711/crayon.jpg',
            'https://s3-us-west-
2.amazonaws.com/s.cdpn.io/175711/spaceship.jpg',
            'https://s3-us-west-
2.amazonaws.com/s.cdpn.io/175711/dj.jpg',
            'https://s3-us-west-
2.amazonaws.com/s.cdpn.io/175711/chicken.jpg'
        ],
*/
        image,
        loaded = 0;
    // very quick and dirty hack to load and display the first
image asap
// дуже швидкий спосіб для завантаження і
відображення першого
// зображення якомога швидше
    images[0] = image = new Image();
        image.onload = function() {
            if (++loaded === 1) {
                imagesLoaded();
                for (var i = 1; i < 4; i++) {
                    images[i] = image = new Image();
                    image.src = urls[i];
                }
            }
        };
        image.src = urls[0];
```

```
};
function imagesLoaded() {
   placeImage(false);
   triangulate();
   shatter();
}

function placeImage(transitionIn) {
   image = images[imageIndex];
   if (++imageIndex === images.length) imageIndex = 0;
// додавання декількох обробчиків на події
   image.addEventListener('click', imageClickHandler);
   container.appendChild(image);
   if (transitionIn !== false) {
      TweenMax.fromTo(image, 0.75, {y:-1000}, {y:0,
ease:Back.easeOut});
   }
}
function imageClickHandler(event) {
   var box = image.getBoundingClientRect(),
      top = box.top,
      left = box.left;
   clickPosition[0] = event.clientX - left;
   clickPosition[1] = event.clientY - top;
   triangulate();
   shatter();
}
function triangulate() {
   var rings = [
         {r:50, c:12},
         {r:150, c:12},
         {r:300, c:12},
         {r:1200, c:12} // very large in case of corner clicks
// дуже великий в разі кутових клацань
      ],
      x,
      y,
      centerX = clickPosition[0],
      centerY = clickPosition[1];
   vertices.push([centerX, centerY]);
   rings.forEach(function(ring) {
      var radius = ring.r,
         count = ring.c,
         variance = radius * 0.25;
```

```
        for (var i = 0; i < count; i++) {
            x = Math.cos((i / count) * TWO_PI) * radius +
centerX + randomRange(-variance, variance);
            y = Math.sin((i / count) * TWO_PI) * radius +
centerY + randomRange(-variance, variance);
            vertices.push([x, y]);
        }
    });

    vertices.forEach(function(v) {
        v[0] = clamp(v[0], 0, imageWidth);
        v[1] = clamp(v[1], 0, imageHeight);
    });

    indices = Delaunay.triangulate(vertices);
}

function shatter() {
    var p0, p1, p2,
        fragment;

    var tl0 =
newTimelineMax({onComplete:shatterCompleteHandler});

    for (var i = 0; i < indices.length; i += 3) {
        p0 = vertices[indices[i + 0]];
        p1 = vertices[indices[i + 1]];
        p2 = vertices[indices[i + 2]];

        fragment = new Fragment(p0, p1, p2);

        var dx = fragment.centroid[0] - clickPosition[0],
            dy = fragment.centroid[1] - clickPosition[1],
            d = Math.sqrt(dx * dx + dy * dy),
            rx = 30 * sign(dy),
            ry = 90 * -sign(dx),
            delay = d * 0.003 * randomRange(0.9, 1.1);
        fragment.canvas.style.zIndex =
Math.floor(d).toString();

        var tl1 = new TimelineMax();
        tl1.to(fragment.canvas, 1, {
            z:-500,
            rotationX:rx,
            rotationY:ry,
            ease:Cubic.easeIn
        });
```

```
        tl1.to(fragment.canvas, 0.4,{alpha:0}, 0.6);

        tl0.insert(tl1, delay);

        fragments.push(fragment);
        container.appendChild(fragment.canvas);
    }

    container.removeChild(image);
    image.removeEventListener('click', imageClickHandler);
}

function shatterCompleteHandler() {
    // add pooling?
    fragments.forEach(function(f) {
        container.removeChild(f.canvas);
    });
    fragments.length = 0;
    vertices.length = 0;
    indices.length = 0;
    placeImage();
}

// MATH UTILS
function randomRange(min, max) {
    return min + (max - min) * Math.random();
}
function clamp(x, min, max)
//тернарні оператори
 {
    return x < min ? min : (x > max ? max : x);
}
function sign(x) {
    return x < 0 ? -1 : 1;
}
//////////////
// FRAGMENT
//////////////

Fragment = function(v0, v1, v2) {
    this.v0 = v0;
    this.v1 = v1;
    this.v2 = v2;
    this.computeBoundingBox();
    this.computeCentroid();
    this.createCanvas();
    this.clip();
};
```

```
Fragment.prototype = {
   computeBoundingBox:function() {
      var xMin = Math.min(this.v0[0], this.v1[0], this.v2[0]),
         xMax = Math.max(this.v0[0], this.v1[0], this.v2[0]),
         yMin = Math.min(this.v0[1], this.v1[1], this.v2[1]),
         yMax = Math.max(this.v0[1], this.v1[1], this.v2[1]);
this.box ={x:xMin, y:yMin,
         w:xMax - xMin,
         h:yMax - yMin
      };
   },
   computeCentroid:function() {
      var x = (this.v0[0] + this.v1[0] + this.v2[0]) / 3,
         y = (this.v0[1] + this.v1[1] + this.v2[1]) / 3;
      this.centroid = [x, y];
   },
   createCanvas:function() {
      this.canvas = document.createElement('canvas');
      this.canvas.width = this.box.w;
      this.canvas.height = this.box.h;
      this.canvas.style.width = this.box.w + 'px';
      this.canvas.style.height = this.box.h + 'px';
      this.canvas.style.left = this.box.x + 'px';
      this.canvas.style.top = this.box.y + 'px';
      this.ctx = this.canvas.getContext('2d');
   },
   clip:function() {
      this.ctx.translate(-this.box.x, -this.box.y);
      this.ctx.beginPath();
      this.ctx.moveTo(this.v0[0], this.v0[1]);
      this.ctx.lineTo(this.v1[0], this.v1[1]);
      this.ctx.lineTo(this.v2[0], this.v2[1]);
      this.ctx.closePath();
      this.ctx.clip();
      this.ctx.drawImage(image, 0, 0);
   }
};

</script>
```

```
body {
 background-color: # 000;
 margin: 0;
 overflow: hidden;
}
```

Фрагмент файлу стилів

```
canvas {
 position: absolute;
 backface-visibility: hidden;
 -webkit-backface-visibility: hidden;
 -moz-backface-visibility: hidden;
 -ms-backface-visibility: hidden;
}

img {
 position: absolute;
 cursor: pointer;
}

#container {
 position: absolute;
 width: 768px;
 height: 485px;
 left: 0;
 right: 0;
 top: 0;
 bottom: 0;
 margin: auto;
}
```

Текст файлу delaunay.js. Реалізація тріангуляції Делоне

```
var Delaunay;

(Function () {
 "Use strict";
 var EPSILON = 1.0 / 1048576.0;
 function supertriangle (vertices) {
 var xmin = Number.POSITIVE_INFINITY,
 ymin = Number.POSITIVE_INFINITY,
 xmax = Number.NEGATIVE_INFINITY,
 ymax = Number.NEGATIVE_INFINITY,
 i, dx, dy, dmax, xmid, ymid;
 for (i = vertices.length; i--;) {
 if (vertices [i] [0] <xmin) xmin = vertices [i] [0];
 if (vertices [i] [0]> xmax) xmax = vertices [i] [0];
 if (vertices [i] [1] <ymin) ymin = vertices [i] [1];
 if (vertices [i] [1]> ymax) ymax = vertices [i] [1];
 }

 dx = xmax - xmin;
 dy = ymax - ymin;
 dmax = Math.max (dx, dy);
 xmid = xmin + dx * 0.5;
 ymid = ymin + dy * 0.5;
```

```
return [
[Xmid - 20 * dmax, ymid - dmax],
[Xmid, ymid + 20 * dmax],
[Xmid + 20 * dmax, ymid - dmax]
];
}

function circumcircle (vertices, i, j, k) {
var x1 = vertices [i] [0],
y1 = vertices [i] [1],
x2 = vertices [j] [0],
y2 = vertices [j] [1],
x3 = vertices [k] [0],
y3 = vertices [k] [1],
fabsy1y2 = Math.abs (y1 - y2),
fabsy2y3 = Math.abs (y2 - y3),
xc, yc, m1, m2, mx1, mx2, my1, my2, dx, dy;

/ * Check for coincident points * /
/ * Перевірка на збіг точок * /
if (fabsy1y2 <EPSILON && fabsy2y3 <EPSILON)
throw new Error ( "Eek! Coincident points! / * Перевірка
на збіг точок * /");

if (fabsy1y2 <EPSILON) {
m2 = - ((x3 - x2) / (y3 - y2));
mx2 = (x2 + x3) / 2.0;
my2 = (y2 + y3) / 2.0;
xc = (x2 + x1) / 2.0;
yc = m2 * (xc - mx2) + my2;
}

else if (fabsy2y3 <EPSILON) {
m1 = - ((x2 - x1) / (y2 - y1));
mx1 = (x1 + x2) / 2.0;
my1 = (y1 + y2) / 2.0;
xc = (x3 + x2) / 2.0;
yc = m1 * (xc - mx1) + my1;
}

else {
m1 = - ((x2 - x1) / (y2 - y1));
m2 = - ((x3 - x2) / (y3 - y2));
mx1 = (x1 + x2) / 2.0;
mx2 = (x2 + x3) / 2.0;
my1 = (y1 + y2) / 2.0;
my2 = (y2 + y3) / 2.0;
```

```
xc = (m1 * mx1 - m2 * mx2 + my2 - my1) / (m1 - m2);
yc = (fabsy1y2> fabsy2y3)?
m1 * (xc - mx1) + my1:
m2 * (xc - mx2) + my2;
}
dx = x2 - xc;
dy = y2 - yc;
return {i: i, j: j, k: k, x: xc, y: yc, r: dx * dx + dy * dy};
}

function dedup (edges) {
var i, j, a, b, m, n;

for (j = edges.length; j;) {
b = edges [- j];
a = edges [- j];

for (i = j; i;) {
n = edges [- i];
m = edges [- i];

if ((a === m && b === n) || (a === n && b === m)) {
edges.splice (j, 2);
edges.splice (i, 2);
break;
}
}
}
}

Delaunay = {
triangulate: function (vertices, key) {
var n = vertices.length,
i, j, indices, st, open, closed, edges, dx, dy, a, b, c;

/ * Bail if there are not enough vertices to form any
triangles. * /
/ * Аванс, якщо не вистачає вершин для формування
будь-яких трикутників. * /
if (n <3)
return [];

/ * Slice out the actual vertices from the passed objects.
(Duplicate the
 * Array even if we do not, though, since we need to make a
supertriangle
 * Later on!) * /
```

```
/ * Вирізати фактичні вершини з пропущених об'єктів. (Дублюйте
 * Масив, навіть якщо ми цього не робимо, тому що нам потрібно зробити супертреугольнік
 * Пізніше!) * /
 vertices = vertices.slice (0);

 if (key)
 for (i = n; i--;)
 vertices [i] = vertices [i] [key];

 / * Make an array of indices into the vertex array, sorted by the
* Vertices 'x-position. Force stable sorting by comparing indices if
* The x-positions are equal. * /
 / * Зробити масив індексів в масив вершин, відсортованого за
* Положення вершин x. Примусове стабільне сортування шляхом порівняння індексів, якщо
* X-позиції рівні. * /

 indices = new Array (n);

 for (i = n; i--;)
 indices [i] = i;

 indices.sort (function (i, j) {
 var diff = vertices [j] [0] - vertices [i] [0];
return diff! == 0? diff: i - j;
 });

 / * Next, find the vertices of the supertriangle (which contains all other
 * Triangles), and append them onto the end of a (copy of) the vertex
* Array. * /

 / * Далі знайдіть вершини супертрикутника (який містить всі інші
 * Трикутники) і додайте їх в кінець (копії) вершини
* Масив. * /

 st = supertriangle (vertices);
 vertices.push (st [0], st [1], st [2]);
```

```
 / * Initialize the open list (containing the supertriangle and
nothing
 * Else) and the closed list (which is empty since we havn't
processed
* Any triangles yet). * /
/ * Ініціалізувати відкритий список (який містить
супертрикутник і нічого * else) і закритий список
(який порожній, адже ми не обробляли * жодних
трикутників поки немає). * /

 open = [circumcircle (vertices, n + 0, n + 1, n + 2)];
 closed = [];
 edges = [];

/ * Incrementally add each vertex to the mesh. * /
/ * Додавати кожну вершину в меш. * /

 for (i = indices.length; i--; edges.length = 0) {
 c = indices [i];

 / * For each open triangle, check to see if the current point
is
* Inside it's circumcircle. If it is, remove the triangle and
add
* It's edges to an edge list. * /
 / * Для кожного відкритого трикутника перевірте, чи
є поточна точка
* Всередині це коло. Якщо це так, видаліть трикутник
і додайте
* Це ребра для списку ребер. * /

 for (j = open.length; j--;) {
 / * If this point is to the right of this triangle's circumcircle,
* Then this triangle should never get checked again.
Remove it
* From the open list, add it to the closed list, and skip. * /

/ * Якщо ця точка знаходиться праворуч від
окружності цього трикутника,
* Тоді цей трикутник ніколи не повинен бути
перевірений знову. Приберіть це
* З відкритого списку, додайте його в закритий список
і пропустіть. * /

 dx = vertices [c] [0] - open [j] .x;
 if (dx> 0.0 && dx * dx> open [j] .r) {
 closed.push (open [j]);
```

```
 open.splice (j, 1);
 continue;
 }

/ * If we're outside the circumcircle, skip this triangle. * /
/ * Якщо ми за межами кола, пропустіть цей
трикутник. * /
 dy = vertices [c] [1] - open [j] .y;
 if (dx * dx + dy * dy - open [j] .r> EPSILON)
 continue;

/ * Remove the triangle and add it's edges to the edge list. * /
/ * Видалити трикутник і додати його ребра в список
ребер. * /

 edges.push (
 open [j] .i, open [j] .j,
 open [j] .j, open [j] .k,
 open [j] .k, open [j] .i
 );
 open.splice (j, 1);
 }

/ * Remove any doubled edges. * // * Видалити всі
подвійні края. * /

 dedup (edges);
/ * Add a new triangle for each edge. * / / * Додати новий
трикутник для кожного ребра. * /
 for (j = edges.length; j;) {
 b = edges [- j];
 a = edges [- j];
 open.push (circumcircle (vertices, a, b, c));
 }
 }
 / * Copy any remaining open triangles to the closed list, and
then
 * remove any triangles that share a vertex with the
supertriangle,
* building a list of triplets that represent triangles. * /
/ * Скопіюйте всі залишені відкриті трикутники в
закритий список, а потім
 * видаліть всі трикутники, які поділяють вершину з
супертрикутником,
* побудова списку триплетів, що являють
трикутники. * /
```

```
 for (i = open.length; i--;)
 closed.push (open [i]);
 open.length = 0;
 for (i = closed.length; i--;)
 if (closed [i] .i <n && closed [i] .j <n && closed [i] .k <n)
 open.push (closed [i] .i, closed [i] .j, closed [i] .k);
/ * Yay, we're done! * /
 return open;
 },
 contains: function (tri, p) {
/ * Bounding box test first, for quick rejections. * /
/ * Спочатку перевіряється обмежувальний
прямокутник, для швидкого відхилення. * /
 if ((p [0] <tri [0] [0] && p [0] <tri [1] [0] && p [0] <tri [2]
[0]) ||
 (P [0]> tri [0] [0] && p [0]> tri [1] [0] && p [0]> tri [2] [0]) ||
 (P [1] <tri [0] [1] && p [1] <tri [1] [1] && p [1] <tri [2] [1]) ||
 (P [1]> tri [0] [1] && p [1]> tri [1] [1] && p [1]> tri [2] [1]))
 return null;
 var a = tri [1] [0] - tri [0] [0],
 b = tri [2] [0] - tri [0] [0],
 c = tri [1] [1] - tri [0] [1],
 d = tri [2] [1] - tri [0] [1],
 i = a * d - b * c;
/ * Degenerate tri. * // * Вироджений три. * /
 if (i === 0.0)
 return null;
 var u = (d * (p [0] - tri [0] [0]) - b * (p [1] - tri [0] [1])) / i,
 v = (a * (p [1] - tri [0] [1]) - c * (p [0] - tri [0] [0])) / i;
/ * If we're outside the tri, fail. * // * Якщо ми поза
тримом, провалимося. * /
 if (u <0.0 || v <0.0 || (u + v)> 1.0)
 return null;
 return [u, v];
 }
 };
 if (typeof module! == "undefined")
 module.exports = Delaunay;
}) ();
```

7.5. Контрольні запитання та завдання

1. Поясніть призначення бібліотеки малювання.
2. Перерахувати методи трансформації CANVAS.
3. Перерахувати методи рисування CANVAS.
4. Перерахувати методи роботи з текстами CANVAS.
5. Поясніть призначення фрази -ctx.beginPath ().
6. Поясніть призначення фрази - ctx.lineTo (e.clientX, e.clientY).
7. Що таке ключове слово -this?
8. Поясніть призначення фрази this.ctx.moveTo (this.v0 [0], this.v0[1]).
9. Поясніть призначення фрази image.addEventListener ('click', imageClickHandler);
10. Що являє інтерфейс Storage?
11. У чому призначення методу setItem () інтерфейсу Storage?
12. У чому призначення формату JSON?
13. Поясніть призначення методу removeEventListener.

7.6. Завдання для самостійного розв’язання

1. Використовуючи методи рисування бібліотеки Canvas, відрисувати квадрат зі сторонами червоного кольору і розмістити його в центрі екрану.

2. Використовуючи код, результат якого представлений на рис. 7.2, домогтися збільшення і зменшення часу рисування, товщини пензля рисування і зміни кольору.

3. Використовуючи код, результат якого наведено на рис. 7.2, внести зміни щоб:

– збільшилася кількість зображень;

– змінився час зміни змісту кожного завантаженого зображення.

8. WEBGL (WEB-BASED GRAPHICS LIBRARY) – КРОСПЛАТФОРМЕННИЙ API ДЛЯ 3D-ГРАФІКИ В БРАУЗЕРІ

8.1. Огляд сучасного стану

WebGL (Web-based Graphics Library) – багатоплатформовий API для 3D-графіки в браузері, що розробляється некомерційною організацією Khronos Group. WebGL використовує мову програмування шейдеров GLSL. WebGL виконується як елемент HTML5, і тому є повноцінною частиною об'єктної моделі документа (DOM API) браузера. Може використовуватися з будь-якими мовами програмування, які вміють працювати з DOM API, наприклад, JavaScript й іншими. Всі провідні розробники браузерів Google (Chrome), Mozilla (Firefox), IApple (Safari), є членами Khronos і реалізують WebGL в своїх браузерах. За рахунок використання низькорівневих засобів підтримки OpenGL частина коду на WebGL може виконуватися безпосередньо на відкритих. WebGL – це контекст елемента canvas HTML, який забезпечує API 3D-графіки без використання плагінів. Перша специфікація була випущена 3 березня 2011 р. Сучасна версія 2.0 (несумісна з версією 1.0) доступна з 27 лютого 2017 року. Дана бібліотека підтримуєтьсяя в браузерах:

– Mozilla Firefox – WebGL був включений на всіх платформах, у яких є потрібна графічна карта з актуальними драйверами, починаючи з версії 4.0.

– Google Chrome – WebGL включений за замовчуванням у всіх версіях, починаючи з 9.

– Safari – експериментально підтримує WebGL, починаючи з версії 5.1, повна підтримка реалізована і включена за замовчуванням у версію 8.0.

– Opera – WebGL реалізований у версії Opera 12.0, але відключений за замовчуванням.

– Internet Explorer – починаючи з Internet Explorer 11 WebGL офіційно підтримується. До виходу 11 версії незалежними розробниками були випущені плагіни Chrome Frame і IEWebGL, що передбачають опції, необхідні для підтримки WebGL в Internet Explorer.

Для спрощення розробки WebGL-додатків існують різні фреймворки і бібліотеки.

Blend4Web дозволяє візуально редагувати контент для WebGL у відкритому пакеті 3D-моделювання та анімації Blender і експортувати його для роботи в браузерах за одну операцію.

Verge3D – WebGL-рендерер, що експортує сцени безпосередньо зі стандартних редакторів (підтримуються Autodesk 3DS MAX і Blender) з можливістю додавання інтерактивних сценаріїв без програмування.

SVG (від англ. Scalable Vector Graphics – Швидка векторна графіка) – мова розмітки, масштабована векторним графіком, створена Консорціумом Всесвітньої мережі (W3C), яка входить до підмножини розширюваної мови розмітки XML, призначена для опису двовимірної векторної і змішаної векторно-растрової графіки в форматі XML. Підтримує як нерухому, так і анімовану інтерактивну графіку або, в інших термінах, декларативну і скриптову. Не підтримує опис тривимірних об'єктів (не плутати з імітацією тривимірності шляхом світлотіні). Це відкритий стандарт, який є рекомендацією консорціуму W3C – організації, яка розробила такі стандарти, як HTML і XHTML. В основу SVG лягли мови розмітки VML і PGML. Розробляється з 1999 року. У 2001 році вийшла версія 1.0, в 2011 – версія 1.1, яка залишається актуальною до сьогоднішнього дня. Нині в активній розробці знаходиться версія 2.

Особливості тривимірної бібліотеки:

– Опис шляхів (англ. path). Дозволяє задати будь-яку фігуру компактним рядком, що описує шлях від початкової точки до кінцевої через будь-які проміжні координати. Рядок з даними задається атрибутом `d` тега `path` і містить команди, закодовані набором літер і чисел. Літера визначає тип команди, числа – її параметри (найчастіше – координати). Команди дозволяють описувати фігури, що складаються з відрізків прямих (L,H,V), кривих Безьє (C,S,Q,T) і дуг (A). Приклад, що описує зірку з 5 ліній, містить рядок даних з командами M (англ. moveto – переміститися до) і L (англ. lineto – нарисувати лінію до), що містять координати точок по X і Y як аргументи. У версіях SVG до 1.2 включно опис шляхів, можливий тільки в пікселях.

– Опис основних геометричних фігур (багатокутники, прямокутники, кола тощо).

– Широкий спектр візуальних властивостей, які можна застосувати до фігур і шляхів: забарвлення, прозорість, обрізка кутів тощо.

– Інтерактивність. На кожен окремий елемент і на ціле зображення можна повісити обробник подій (клік, переміщення, натискання клавіші і т.д), таким чином, користувач може управляти малюнком (наприклад, переміщувати мишкою деякі елементи).

– Анімація і сценарії. За допомогою ECMAScript або JavaScript_можна описувати навіть найскладніші сценарії, пов’язані з математичними обчисленнями координат і пропорцій фігур. Разом з інтерактивністю і SMIL-анімацією це дає дуже широкі можливості для розробників веб-графіки.

– Текстовий формат. Файли SVG можна читати і редагувати (за наявності деяких навичок) за допомогою звичайних текстових редакторів. Під час перегляду документів, що містять SVG-графіку, є доступ до перегляду коду файла, і можливість збереження всього документа. Крім того, SVG-файли зазвичай виходять менші за розміром, ніж порівняні за якістю зображення в форматах JPEG або GIF, а також добре піддаються стисненню.

– Масштабованість. SVG є векторним форматом. Існує можливість збільшити будь-яку частину зображення SVG без втрати якості. Додатково, до елементів SVG-документа можна використовувати фільтри – спеціальні модифікатори для створення ефектів, подібних застосовуваним в ході обробки растрових зображень (розмиття, видавлювання, складні системи трансформації та ін.). У тексті SVG-коду фільтри описуються тегами, візуалізацію яких забезпечує засіб перегляду, що не впливає на розмір вихідного файла, забезпечуючи при цьому необхідну ілюстративну виразність.

– Широким є використання растрової графіки в SVG-документах. Є можливість вставляти елементи з зображеннями в форматах PNG, GIF або JPG.

– Текст у графіці SVG є текстом, а не зображенням, тому його можна виділяти і копіювати, він індексується пошуковими машинами, не потрібно створювати додаткові метафайли для пошукових роботів.

– Анімація реалізована у SVG за допомогою мови SMIL (Synchronized Multimedia Integration Language), розробленої також консорціумом W3C. Підтримуються скриптові мови на основі специфікації ECMAScript. SVG-елементами можна керувати за допомогою JavaScript. Застосування скриптової анімації в SVG дозволяє створювати динамічну й інтерактивну графіку. У SVG забезпечується подієва модель, відстежуються події (завантаження сторінки, зміна її параметрів, події мишки, клавіатури та ін.) Анімація може запускатися за певної події (наприклад, «onmouseover» або «onclick»), що надає графіку інтерактивність. У кожного елемента є свої власні події, до яких можна прив’язувати окремі скрипти.

– SVG – відкритий стандарт. На відміну від деяких інших форматів, SVG не є чиєюсь власністю.

– SVG-документи легко інтегруються з HTML- і XHTML-документами. Зовнішні SVG підключаються через тег **<object>**, значення атрибута data – ім'я файла з розширенням «.svg», що містить розмітку SVG, type – MIME-тип, тобто image / svg + xml. Атрибути width і height визначають розміри області SVG по горизонталі і по вертикалі. Елементи SVG сумісні з HTML і DHTML.

– Сумісність з CSS (англ. Cascading Style Sheets). Відображенням (форматуванням і декоруванням) SVG-елементів можна управляти за допомогою таблиці стилів CSS 2.0 і її розширень, або безпосередньо за допомогою атрибутів SVG-елементів.

SVG надає всі переваги XML:

– Можливість роботи в різних середовищах.

– Інтернаціоналізація (підтримка Юнікода).

– Широка доступність для різних додатків.

– Легка модифікація через стандартні API – наприклад, DOM. SVG підтримує стандартизовану W3C об'єктну модель документа DOM, забезпечуючи доступ до будь-якого елемента, що дає широкі можливості з динамічною зміною елементів, їх атрибутів і подій.

– Легке перетворення таблицями стилів XSLT. Як будь-який заснований на XML формат, SVG дає можливість використовувати для його обробки таблиці трансформації (XSLT). Перетворюючи XML-дані в SVG за допомогою простого XSL, є можливість легко отримати графічне подання будь-яких даних, наприклад, візуалізацію хімічних молекул, описаних мовою CML.

Недоліки формату:

– Повністю відсутня підтримка тривимірної графіки SVG успадковує всі недоліки XML, такі як великий розмір файла (втім, останній компенсується існуванням стисненого формату SVGZ, проте його використання на даний момент ускладнено, оскільки SVGZ не має власного mime-type).

– Складність використання у великих картографічних додатках через те, що для правильного відображення маленької частини зображення документ необхідно прочитати цілком.

– Чим більше в зображенні дрібних деталей, тим швидше зростає розмір SVG-даних. Граничний випадок – коли зображення є білий шум. У цьому випадку SVG не тільки не дає жодних переваг у розмірі файлу, але навіть програє відносно растровому формату. На практиці SVG стає невигідним вже задовго до того, як зображення дійде до стадії білого шуму.

8.2. Опис структури бібліотеки Three.js

Three.js – легка кросбраузерна бібліотека JavaScript, що використовується для створення та відображення анімованої комп'ютерної 3D-графіки під час розробки веб-додатків. Three.js-скрипти можуть використовуватися спільно з елементом HTML5 CANVAS, SVG або WebGL. Вихідний код розташований в репозиторії сайту GitHub.

Three.js дозволяє створювати прискорену на GPU 3D-графіку, використовуючи мову JavaScript як частину сайту без підключення пропрієтарних плагінів для браузера. Це можливо завдяки використанню технології WebGL.

У бібліотеці використовуються рендерер: Canvas, SVG або WebGL.

Сцена дозволяє робити: додавання і видалення об'єктів у режимі реального часу; туман.

Камери бувають: перспективна або ортографічна.

Анімація складається з: каркасів, прямої кінематики, інверсної кінематики, покадрової анімації.

Джерела світла бувають: зовнішнє, спрямоване, точкове; тіні: кинуті та отримані.

Шейдери дають повний доступ до всіх OpenGL-шейдерам (GLSL).

Об'єктами є: мережі, частинки, спрайт, лінії, скелетна анімація тощо.

Геометрією є: площина, куб, сфера, тор, 3D-текст тощо.

Модифікатори: тканина, видавлювання.

Завантажники даних: двійковий, зображення, JSON і сцена.

Експорт та імпорт: утиліти, що створюють Three.js-сумісні JSON-файли з форматів Blender, openCTM, FBX, 3D-Studio Max, і Wavefront .obj-файл.

Підтримка: документація з API-бібліотеки знаходиться в процесі постійного розширення і доповнення, є публічний форум і велике співтовариство.

Приклади: на офіційному сайті можна знайти понад 150 прикладів роботи зі шрифтами, моделями, текстурами, звуком та іншими елементами сцени.

Бібліотека Three.js працює в усіх браузерах, які підтримують технологію WebGL; також може працювати з «чистим» інтерфейсом елемента CANVAS, завдяки чому працює і на багатьох мобільних пристроях. Three.js поширюється під ліцензією MIT license.

Three.js – це потужний інструмент. Він допомагає використовувати 3D-дизайн у браузері з прийнятною продуктивністю. Початок Three.js може бути складним, особливо якщо ви ніколи не занурювалися в світ 3D-програмування раніше.

Основні поняття, необхідні для створення тривимірних зображень, наведені на рис.8.1 та 8.2.

1. Ренедерер – сутність, яка відображає на canvas обрану сцену з обраної камери.

2. Сцена-3d – простір, в якому розташовуються джерела світла і елементи-«меши».

3…..

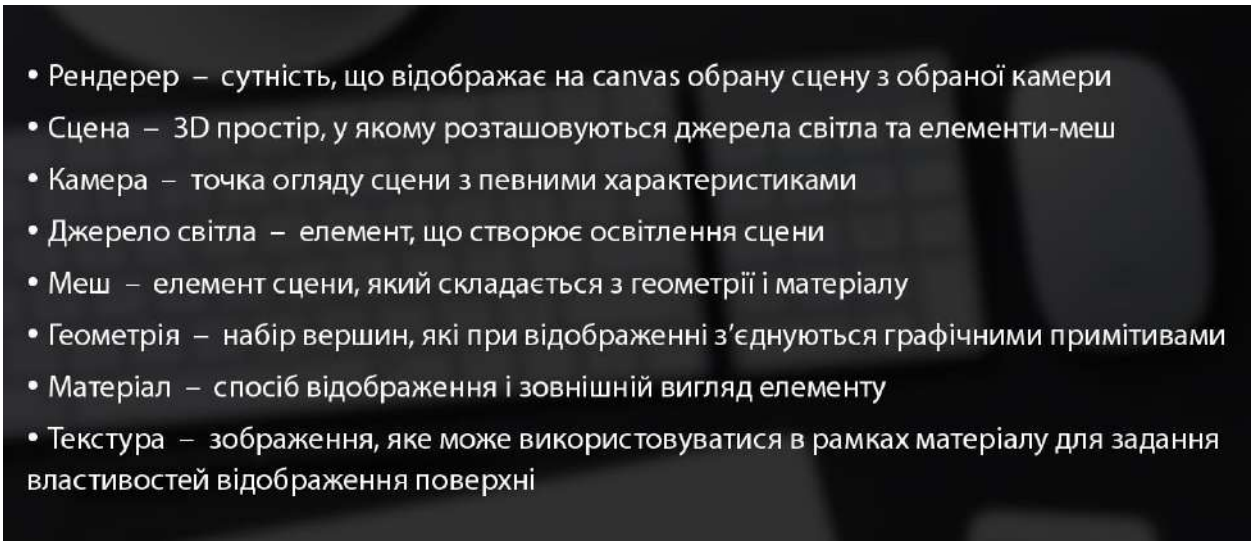


Рисунок 8.1 – Основні поняття бібліотеки тривимірної графіки three.js

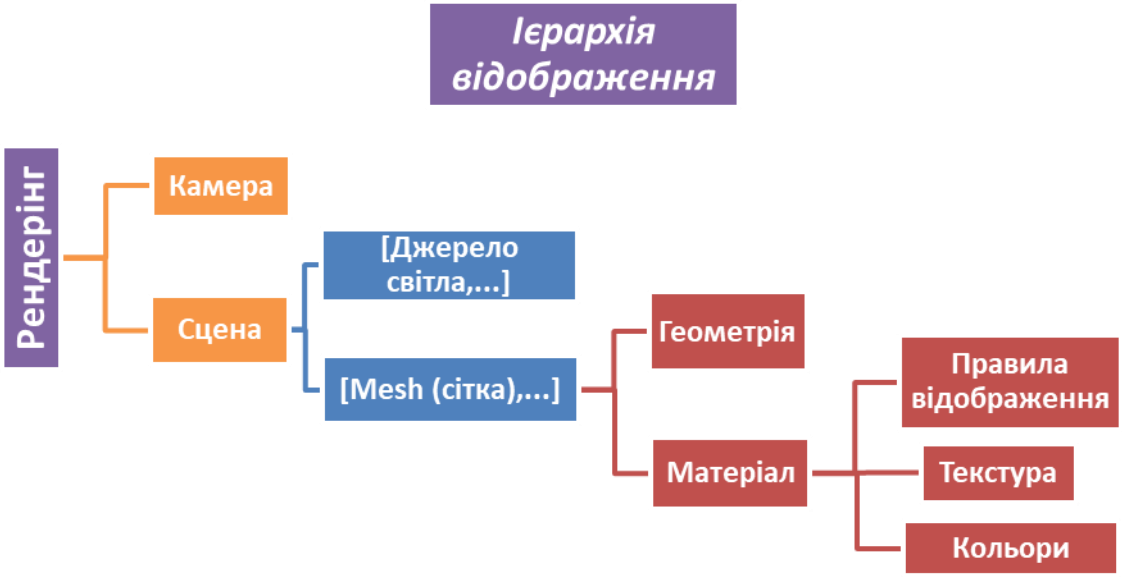


Рисунок 8.2 – Схеми ієрархії відображень бібліотеки three.js

8.2.1. Вектори і контейнери – основні будівельні блоки бібліотеки

Найчастіше виділяють два основних класи в Three.js – *Vector3* і *Box3*. Якщо ви новачок у 3D, то це може звучати дещо абстрактно, але ви ще зустрінете їх дуже багато разів.

Vector3 – це найголовніший 3D-клас, що містить три числа: x, y і z. Числа є координатами точки в 3D-просторі (або напрямок і довжина).

Наприклад: const vect = new THREE.Vector3 (1, 1, 1).

Велика частина конструкторів у Three.js приймають об'єкти типу *Vector3* як вхідні аргументи. Наприклад, *Box3*.

Box3 – цей клас представляє кубойд (3d-контейнер). Його головне завдання – створити контейнер навколо інших об'єктів – і все. Найменший кубойд, в який поміститься 3D-об'єкт. Кожен *Box3* вирівнюється по осях x, y і z.

Приклад, як створити контейнер, використовуючи *Vector3*:

```
const vect = new THREE.Vector3 (1, 1, 1);
const box = new THREE.Box3 (vect);
Приклад як створити контейнер навколо вже наявного 3D об'єкта:
const box = new THREE.Box3 ();
box.setFromObject (object);
```

Можна створювати сітки і без цих глибоких знань, але як тільки ви почнете придумувати або змінювати свої моделі, ці класи точно знадобляться. Зараз ми підемо від абстракцій до більш видимих речей.

Полігональна сітка

У Three.js основний візуальний елемент на сцені – Mesh. Це 3D-об'єкт, складений з трикутних прямокутників (полігональних сітка). Він будується за допомогою двох об'єктів:

Geometry – визначає його форму, *Material* – зовнішній вигляд.

Їх визначення можуть здатися дещо заплутаними (наприклад, клас *Geometry* може містити інформацію про колір), але головна відмінність саме така.

Geometry

Грунтуючись на завдання, яке ви хочете виконати, можливо вам захочется визначити геометрію всередині Three.js або імпортувати іншу з файлу. Використовуючи такі функції, як THREE. *TorusKnotGeometry*, можемо створити складні об'єкти одним рядком коду.

Найпростіша 3D-фігура, кубойд або контейнер, може бути заданий параметрами *width, height і depth*.

Геометрія задається як:

const geometry = new THREE. BoxGeometry (20, 20, 20);

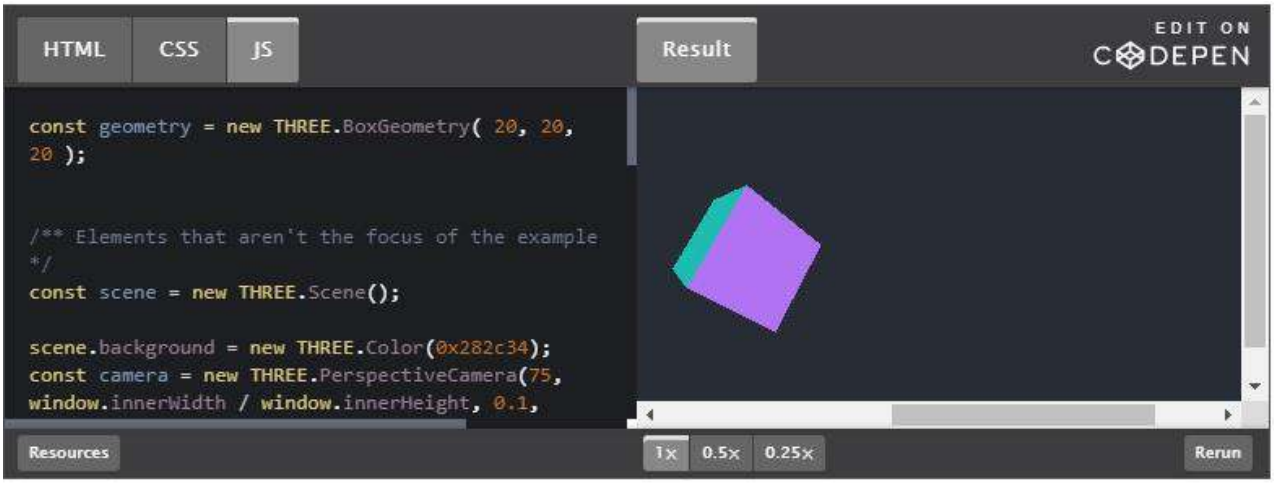


Рисунок 8.3 – Приклад задання об’єкта за допомогою three.js

Для сфери мінімально потрібно значення парамерів radius, widthSegments і heightSegments. Дві останні змінні вказують, скільки трикутників у моделі має використовуватися, щоб зобразити сферу: більша кількість – більш гладкою виглядатиме.

Приклад задання геометрії:

const geometry = new THREE.SphereGeometry (20, 64, 64).

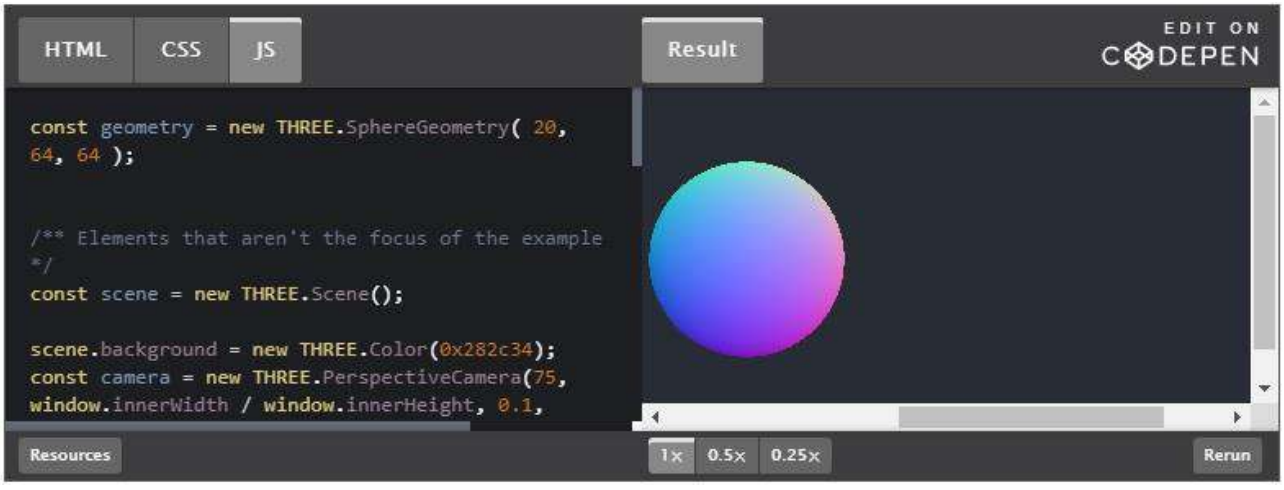


Рисунок 8.4 – Схеми ієрархії відображень бібліотеки three.js

Якщо ми хочемо зробити гострі або трикутні форми, то можна використовувати конус. Його аргументи – це поєднання аргументів двох попередніх фігур. Нижче ми прописуємо *radius, widthSegments і radialSegments.*

Приклад задання конуса:

const geometry = new THREE.ConeBufferGeometry (5, 20, 32).

Це лише частина найпоширеніших фігур. Three.js має всередині дуже багато фігур, які можна подивитися в документації. Розглянемо більш цікаві форми, побудовані на основі методу *TorusKnotGeometry.*

Виклик методу здійснюється так:

const geometry = new THREE.TorusKnotGeometry (10,1.3,500,6,6,20).

Приклад використання на

сайтіhttps://codepen.io/BretCameron/pen/gOYqORg.

8.2.2. Матеріали

Геометрія задає форму наших 3D-об'єктів, але не їх зовнішній вигляд. Щоб це виправити, нам потрібні матеріали.

Three.js пропонує десять видів матеріалів, кожен з який має свої плюси й налаштування параметрів. Ми розглянемо лише частину найкорисніших матеріалів.

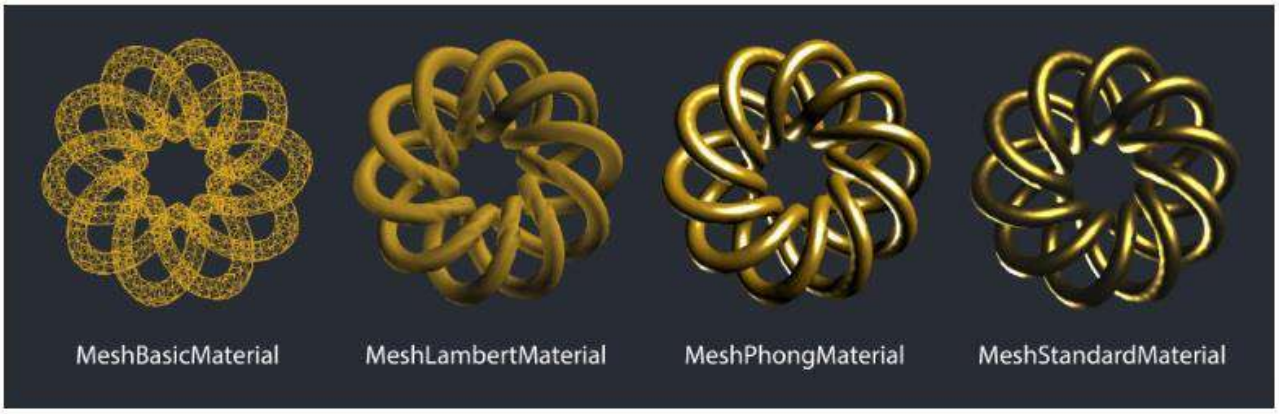


Рисунок 8.5 – Приклади використання властивостей матеріалу в three.js

Матеріал MeshNormalMaterial корисний при швидкому старті й запуску. MeshNormalMaterial – це багатобарвний матеріал, який ми використовували в прикладах вище. Він відповідає нормальним векторам в панелі RGB, іншими словами, використовуються кольори для визначення позиції вектора в 3D-просторі const material = new THREE. MeshNormalMaterial ().

Зауважимо, що якщо ви хочете поміняти колір матеріалу, то можна використовувати CSS-фільтр і змінювати насиченість: filter: hue-rotate (90deg). Цей матеріал більш корисний для швидкого старту і запуску.

Матеріал MeshBasicMaterial корисний під час відображення тільки скелета. Якщо ви хочете надати фігурі єдиний колір, то можна використовувати MeshBasicMaterial, тільки якщо не застосовується освітлення. Корисно застосування того матеріалу в відображенні скелета моделі. Для відтворення тільки скелета потрібно передати {wireframe: true} як параметр.

const material = new THREE.MeshBasicMaterial ({wireframe: true, color: 0xdaa520})

Головний недолік цього матеріалу в тому, що абсолютно зникає інформація про глибину матеріалу. Кожен матеріал має опцію для відображення тільки скелета, але лише один матеріал вирішує проблему відсутність глибини – MeshDepthMaterial.

MeshLambertMaterial корисний за високої продуктивності, але низької точності. Це перший матеріал, який враховує освітленість, тому потрібно додати трохи світла на нашу сцену. У коді нижче ми додаємо точкові джерела світла з жовтим відтінком для створення більш теплого ефекту:

```
const scene = new THREE.Scene ();
const frontSpot = new THREE.SpotLight (0xeeeece);
frontSpot.position.set (1000 1000 1000);
scene.add (frontSpot);
const frontSpot2 = new THREE.SpotLight (0xdddce);
frontSpot2.position.set (-500, -500, -500);
scene.add (frontSpot2).
```

Тепер додамо матеріал для нашої фігури. Оскільки фігура схожа на прикрасу, пропонується додати більш золотистий колір. Інший параметр, emissive, – це колір об’єкта, виходить від самого об’єкта (без джерела світла).

Як можна побачити в прикладі нижче, колір більш-менш правильний, але те, як він взаємодіє зі світлом, не додає реалістичності.

Розглянемо методи MeshPhongMaterial або MeshStandardMaterial.

Цей материал використовують за середньої продуктивності і середньої точності. Він пропонує компроміс між продуктивністю і точністю відтворення, тому цей матеріал – гарний варіант для додатка, який має бути продуктивним поряд з більш точною відрисовкою, ніж при MeshLambertMaterial.

Зараз ми можемо змінювати властивість specular, яка впливає на яскравість і колір відображення поверхні. Якщо властивість emissive зазвичай темна, то specular краще працює для світлих кольорів.

Нижче ми використовуємо світлий сірий:

```
const material = new THREE.MeshPhongMaterial ({
 color: 0xdaa520,
 emissive: 0x000000,
 specular: 0xbcbcbc,
}).
```

Візуально зображення зверху відбиває світло більш переконливо, але все ще не ідеально. Біле світо занадто яскраве, і матеріал виглядає більш ребристо, ніж металево (а ми прагнемо саме до цього). Ми можемо отримати кращий результат, використовуючи MeshStandardMaterial.

MeshStandartMaterial корисний за високої точності, але низької продуктивності. Це найточніший матеріал з усіх, хоча його використання спричинить більші витрати. MeshStandartMaterial використовується з додатковими параметрами metalness і roughness, кожен з яких набуває значення між 0 і 1. Параметр metalness впливає на те, як об'єкт відбивае світло, стаючи ближче природі металу. Все тому що провідникові матеріали як метали мають інші властивості на відміну від діелектриків, таких як кераміка.

Roughness додає додатковий шар для кастомізації. Можна уявити його як протилежність глянсовості: 0 – дуже глянсовий, 1 – дуже матовий.

Виклик матеріалу здійснюється:

```
const material = new THREE. MeshStandardMaterial ({ color: 0xfcc742, emissive:
0x111111, specular: 0xffffff, metalness: 1, roughness: 0.55,});
```

Це найреалістичніший матеріал з усіх поданих в Three.js, але і найресурсозатратніший.

8.2.3. Завантажники

Як ми вже обговорили вище, можна вручну визначати геометрію і полігональні сітки. На практиці розробники частіше завантажують свої геометрії з файлів. На щастя, three.js має трохи підтримуваних завантажників, які підтримують багато 3D-форматів.

Основний ObjectLoader завантажує JSON-файл, використовуючи JSON Object / Scene format. Більшість завантажників потрібно імпортувати вручну. Можна знайти повний список підтримуваних завантажників тут, та імпортувати їх. Нижче невеликий список того, що можна імпортувати

```
// Import dependencies
import * as THREE from 'three';
```

```
import {OrbitControls} from 'three / examples / jsm / controls / OrbitControls.js';
// GLTF
import {GLTFLoader} from 'three / examples / jsm / loaders / GLTFLoader.js';
// OBJ
import {OBJLoader} from 'three / examples / jsm / loaders / OBJLoader.js';
// STL
import {STLLoader} from 'three / examples / jsm / loaders / STLLoader.js';
// FBX
import {FBXLoader} from 'three / examples / jsm / loaders / FBXLoader.js';
// 3MF
import {3MFLoader} from 'three / examples / jsm / loaders / 3MFLoader.js'.
```

Рекомендований формат для онлайн-перегляду – GLTF, через те, що формат «спрямований на доставку Ассет в Рантайм, компактний для передачі і швидкий для завантаження».

Звісно, може бути дуже багато причин віддавати перевагу певному типу файлів (наприклад, якщо якість у пріоритеті або потрібна точність для 3D-друку). Краща ж продуктивність онлайн буде з імпортом GLTF.

```
import {GLTFLoader} from 'three / examples / jsm / loaders / GLTFLoader.js';
import model from '../models/sample.gltf';
let loader = new GLTFLoader ();
loader.load (model, function (geometry) {
 // if the model is loaded successfully, add it to your scene here
}, Undefined, function (err) {
 console.error (err);
}).
```

Одна з причин, чому Three.js може створити щось з нуля, – лише пара рядків коду. У кожному прикладі вище нам потрібно було створити сцену і камеру. Щоб спростити, ми тримали цей код за рамками розгляду, але зараз подивимося, як це виглядатиме все разом. Те, як ви організуєте свій код, вирішуєте тільки ви. У простіших прикладах, таких як у цьому пункті, є сенс писати весь код в одному місці. Але на практиці корисно розділяти окремі елементи для можливості розширення кодової бази та її управління. Для простоти розглянемо елементи, які відрисовуються як один об’єкт, тому весь код ми розмістимо в одному файлі. Нижче наведені приклади імпорту файлів:

```
// Import dependencies
import * as THREE from 'three';
import {OrbitControls} from 'three / examples / jsm / controls / OrbitControls.js';
// Import dependencies
import * as THREE from 'three';
```

```
import {OrbitControls} from 'three / examples / jsm / controls / OrbitControls.js';
// Створюємо сцену
const scene = new THREE.Scene ();
scene.background = new THREE.Color (0x282c34);
// Визначаємо камеру, встановлюємо її на заповнення вікна браузера
const camera = new THREE.PerspectiveCamera (75, window.innerWidth /
window.innerHeight, 0.1, 10000);
camera.position.z = 5;
//Встановлюємо "рисувальника" і встановлюємо на вікно браузера
const renderer = new THREE.WebGLRenderer ();
renderer.setSize (window.innerWidth, window.innerHeight);
// Беремо елемент DOM і прикріплюємо renderer.domElement до нього
document.getElementById ( 'threejs'). appendChild (renderer.domElement);
// Додаємо управління, встановлюємо як мету той самий DOM елемент
let controls = new OrbitControls (camera, document.getElementById ( 'threejs'));
controls.target.set (0, 0, 0);
controls.rotateSpeed = 0.5;
controls.update ();

// Визначаємо (або імпортуємо) геометрію об'єкта
const geometry = new THREE.TorusKnotGeometry (10, 1.3, 500, 6, 6, 20);
// Визначаємо матеріал об'єкта
const material = new THREE.MeshStandardMaterial ({
 color: 0xfcc742,
 emissive: 0x111111,
 specular: 0xffffff,
 metalness: 1,
 roughness: 0.55,
});
// Створюємо полігональну мережу, масштабуємо її і додаємо на сцену
const mesh = new THREE.Mesh (geometry, material);

mesh.scale.x = 0.1;
mesh.scale.y = 0.1;
mesh.scale.z = 0.1;

scene.add (mesh);

// Додаємо освітлення, встановлюємо його і додаємо на сцену
const frontSpot = new THREE.SpotLight (0xeeeece);
const frontSpot2 = new THREE.SpotLight (0xddddce);
frontSpot.position.set (1000 1000 1000);
frontSpot2.position.set (-500, -500, -500);
scene.add (frontSpot);
scene.add (frontSpot2);
```

```
// Створюємо функцію анімації, яка дозволить відрисовувати вашу сцену і визначити будь-який рух
const animate = function () {
 requestAnimationFrame (animate);
 mesh.rotation.x + = 0.005;
 mesh.rotation.y + = 0.005;
 mesh.rotation.z + = 0.005;
 renderer.render (scene, camera);
};
// Викликаємо функцію анімації
animate ();
```

Використання фреймворк

Зараз існує гарний пакет *react-three-fiber* для React. Для користувачів React є очевидні переваги користування пакетом – зберігається структура роботи з компонентами, яка дозволяє перевикористати код. Для новачків краще почати зі звичайного *Vanila JS,* оскільки більшість онлайн-матеріалів, написаних про Three.js, належать до Three.js на Vanila JS. Може бути заплутано і важко їх вивчати через пакет. Наприклад, вам доведеться транслювати Three.js-об'єкти і методи на компоненти. (Як тільки ви опануєте Three.js, можете використовувати будь-який пакет).

Додавання Three.js у фреймворк

Three.js дає HTML об'єкт (найчастіше називається він *renderer.domElement*) який може бути доданий до будь-якого HTML-об'єкту в вашому додатку. Наприклад, якщо у вас є *div з id = "three.js*" ви можете просто додати наступний код у ваш Three.js-код:

```
document.getElementById ( 'threejs'). appendChild (renderer.domElement).
```

Деякі фреймворки мають кращі шляхи звернення до вузлів DOM-дерева. Наприклад, ref в React, $ ref у Vue або ngRef у Angular, і це великий плюс на тлі прямого звернення до елементів DOM.

Як приклад, давайте розглянемо швидку реалізацію для React.

Стратегія для модіфікованой бібліотеки – React

Якщо ви використовуєте фреймверк (бібліотеку) React.js, то існує один шлях впровадження Three.js-файлів в один з ваших компонентів. У файлі ThreeEntryPoint.js ми напишемо такий код:

```
export default function ThreeEntryPoint (sceneRef) {
 let renderer = new THREE.WebGLRenderer ();
```

```
 // ...
 sceneRef.appendChild (renderer.domElement);
}.
```

Ми експортуємо це як функцію, яка приймає один аргумент: посилання на елемент в нашому компоненті. Тепер ми можемо створити наш компонент:

```
import React, {Component} from 'react';
import ThreeEntryPoint from './threejs/ThreeEntryPoint';
export default class ThreeContainer extends Component {
componentDidMount () {
   ThreeEntryPoint(this.scene);
  }
render() {
   return (
     <>
       <div ref={element => this.scene = element} />
     </>
   );
  }
}
```

Імпортована функція ThreeEntryPoint має викликатися в методі *componentDidMount* і передавати новий *div* як аргумент, використовуючи посилання. Як приклад такого підходу в дії можна склонувати репозиторій і спробувати самостійно: https://github.com/BretCameron/three-js-sample.

Наступний приклад наочно демонструє можливості бібліотеки.

```
<! DOCTYPE html>
<html>
<head>
        <title> </ title>
        <meta charset = "utf-8">
<meta name = "viewport" content = "width = device-width,
initial-scale = 1">

<link type = "text / css" rel = "stylesheet" href =
"pointsphere.css">
<script src = "three.js"> </ script>
<script src = "dat.gui.min.js"> </ script>

</ head>
<style type = "text / css">
html, body
{margin: 0;
overflow: hidden;
}
```

```
</ style>
<body>
<canvas id = "canvas"> </ canvas>
<script>

window.onload = function () {
var width = window.innerWidth;
var height = window.innerHeight;
var canvas = document.getElementById ( 'canvas');
canvas.setAttribute ( 'height', height);
canvas.setAttribute ( 'width', width);

// змінна бал додається в останній момент
var ball = {rotationY: 0,
rotationX: 0,
rotationZ: 0,
positionX: 0,
positionY: 0,
positionZ: 0,

};
// gui add with змінна бал додається в останній момент
// var gui = new DATGUI ();
var gui = new dat.GUI ();
gui.add (ball, 'rotationY'). min (-5) .max (1) .step (0.001);
/ *
gui.add (ball, 'rotationX'). min (-5) .max (0.2) .step. (0.001);
gui.add (ball, 'rotationZ'). min (-0.2) .max (0.2) .step. (0.001);
* /
gui.add (ball, 'positionX'). min (-0.2) .max (2) .step (0.01);
/ *
gui.add (ball, 'positionY'). min (-0.2) .max (0.2) .step. (0.001);
gui.add (ball, 'positionZ'). min (-0.2) .max (0.2) .step. (0.001);
* /
var renderer = new THREE.WebGLRenderer ({canvas:
canvas});
 renderer.setClearColor (0x000000);
var scene = new THREE.Scene ();
var camera = new THREE.PerspectiveCamera (45, width /
height, 0.1, 5000);
camera.position.set (100,0,1000);
var light = new THREE.AmbientLight (0xffffff);
var geometry = new THREE.SphereGeometry (200,12,12);
// wireframe: true-забезпечує порожнє тіло скелет .... сфери
// var material = new THREE.MeshBasicMaterial ({color:
0x00ff00, wireframe: true});
```

	// {vertexColor: THREE.FaceColors} заповнює межі колір скелет зникне сфера зникне буде плавати коло. *var material = new THREE.MeshBasicMaterial ({color: 0xf0afff, vertexColors: THREE.FaceColors});* *// цикл забезпечує заповнення граней сфери відтінками три випадкових параметра* *for (var i = 0; i <geometry.faces.length; i ++) {* *geometry.faces [i] .color.setRGB (Math.random (), Math.random (), Math.random ());* *}* *var mesh = new THREE.Mesh (geometry, material);* *scene.add (mesh)* *//renderer.render(scene, camera);* *var animate = function () {* *requestAnimationFrame (animate);* *// завдання параметрів анімації-обертання і переміщення* *//mesh.position.x=mesh.position.x+0.005;* *mesh.position.x + = 0.005;* *mesh.rotation.y + = 0.005;* *// mesh.rotation.z + = 0.005;* *//mesh.position.x=mesh.position.x+0.005;* *//* *mesh.position.x + = ball.positionX;* *// mesh.position.y + = ball.positionY;* *// mesh.position.z + = ball.positionZ;* *//mesh.rotation.x + = ball.rotationX;* *mesh.rotation.y + = ball.rotationY;* *// mesh.rotation.z + = ball.rotationZ;* *// * /* *renderer.render (scene, camera);* *};* *animate ();* *}* *</ script>* *</ body>* *</ html>*
Структура файла pointsphere.css	*body* *{margin: 0; }* *canvas* *{width: 100%;* *height: 100%}*

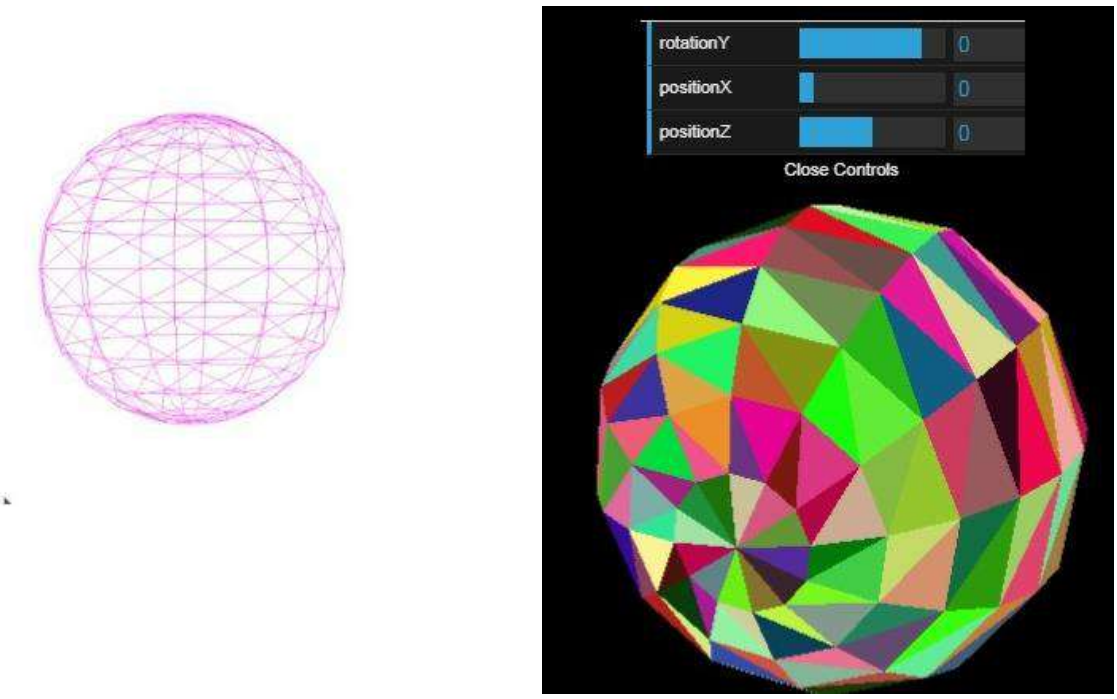


Рисунок 8.6 – Результати фрагментів виконання коду

8.3. Контрольні запитання та завдання

1.У чому призначення бібліотеки three.js?
2. Назвіть основні поняття і поясніть структуру бібліотеки three.js.
3. Якою функцією задається геометрія в бібліотеці three.js?
4. Які види матеріалів використовуються в структурі бібліотеки?
5. Наведіть приклад використання завантажників геометрії і матеріалів.
6. У чому призначення функції OrbitControls.js?
7. У чому призначення функції OBJLoader.js?
8. Яка функція запускає анімацію в коді, наведеному у табл. 8.1?
8. У чому полягає призначення бібліотеки dat.gui.min.js?

8.4. Завдання для самостійного розв’язання

1. З сайту github скачайте файл (бібліотеку) three.js і dat.gui.min.js .
2. Скопіюйте в редактор коду текст, наведений у табл. 8.1, розмістіть його в каталозі з файлами three.js idat.gui.min.js і збережіть з розширенням * .html.
3. Відкрийте код, який реалізує сюжет, наведений на рис. 8.2, в браузері і переконайтеся в його працездатності.
4. По черзі прибираючи коментарі в коді з табл. 8.1, зробіть такі зміни:
– переміщення сфери по осях x, y, z;
– кольори сфери;
– швидкості обертання сфери.

9. ОСНОВИ ПОЛІГОНАЛЬНОГО МОДЕЛЮВАННЯ У 3DS MAX

3D графіка – розділ комп'ютерної графіки, що спеціалізується на побудові геометричної проєкції тривимірної моделі сцени на площину (наприклад, екран комп'ютера) за допомогою спеціалізованих програм.

Завдання 3D-моделювання – розробка візуального об'ємного образу бажаного об'єкта. Отримана модель може відповідати об'єктам реального світу (наприклад, будівля, людина, автомобіль) або бути цілком абстрактною (проєкція чотиривимірного фрактала).

9.1. Полігональне моделювання

Полігональне моделювання є одним з основних способів моделювання. Суть в тому, що всі поверхні уявляються у вигляді простих геометричних двовимірних примітивів. Редагування сітки або редагування каркаса об'єкта є низькорівневим моделюванням і базоване на маніпулюванні з вершинами, ребрами і гранями об'єктів.

Полігональна сітка (Polygon mesh) – це сукупність вершин, ребер і граней, які визначають форму багатогранного об'єкта у тривимірній комп'ютерній графіці. Таким чином, кожен об'єкт можна уявити у вигляді набору найпростіших полігонів (рис. 9.1).

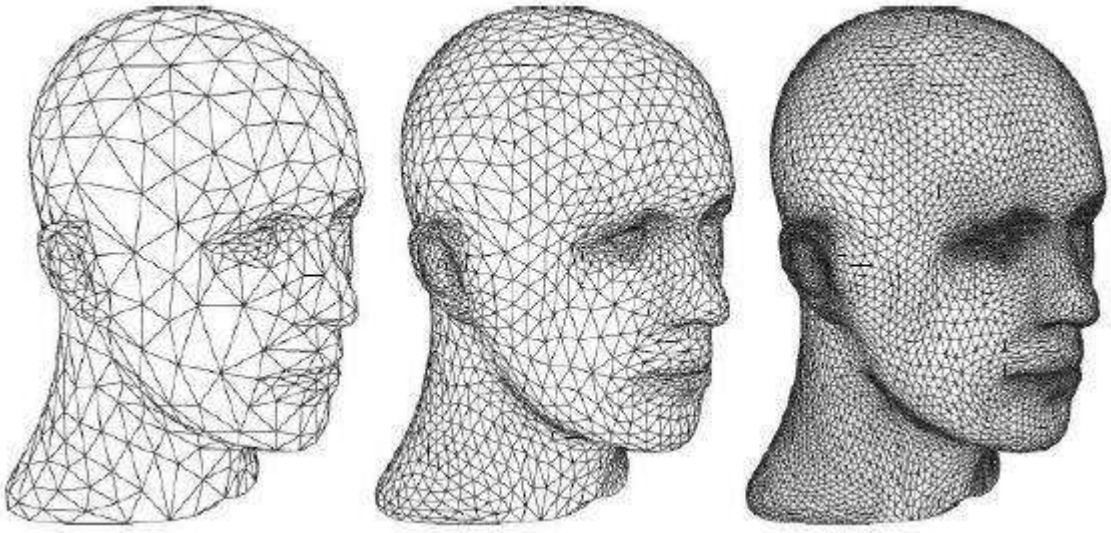


Рисунок 9.1 – Полігональна сітка моделі

При полігональному моделюванні змінюють форму об'єкта, безпосередньо впливаючи на його складові, такі як:

Вершина (Vertex) – точка, піддається редагуванню, присвоюванню формул взаємодії (так званих *ваг*) з прилеглими кривими.

Ребро (Border) – пряма, що з'єднує дві вершини.

Грань – це замкнута безліч ребер, в якій трикутна грань має три ребра і чотирикутна – чотири.

Полігон – мінімальна поверхня для візуалізації. Оскільки в багатьох задачах комп'ютерної графіки потрібно, щоб об'єкт був опуклим багатокутником, як мінімальний полігон зазвичай застосовують трикутники.

Теселяція – автоматизований процес додавання нових опуклих багатокутників у полігональну сітку з метою підвищення деталізації сітки.

Примітиви – найпростіший геометричний об'єкт, що формується в програмі, наприклад, це куб, куля (рис. 9.2), площина, дуга і т. д. Тривимірні примітиви становлять основу багатьох програмних пакетів комп'ютерної графіки і забезпечують можливість створення різноманітних об'єктів простої форми. Часто для формування потрібної моделі тривимірні примітиви доводиться об'єднувати або модифікувати.

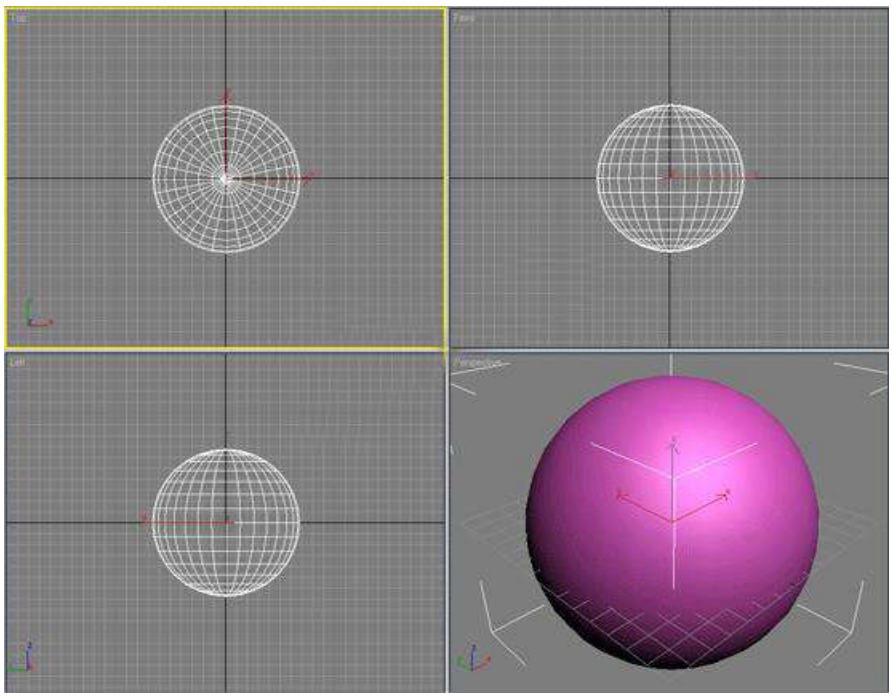


Рисунок 9.2 – Примітив типу Куля

Один з найбільш ефективних способів модифікації об'єктів – редагування їх на рівні підоб'єктів, якими можуть бути вершини, ребра, грані, сегменти, сплайни в цілому та ін. Такий підхід дозволяє коригувати складні об'єкти, що складаються з сукупності більш простих.

Модифікатором називається дія, яка спрямована на об’єкт, в результаті чого властивості об’єкта змінюються. Наприклад, модифікатор може діяти на об’єкт, деформуючи його різними способами – згинаючи, витягаючи, скручуючи і т. ін. Модифікатор також може служити для керування становищем текстури на об’єкті або змінювати фізичні властивості об’єкта, наприклад, робити його гнучким.

9.2. Робота зі сплайнами

Сплайн – крива, що з’єднує дві і більше вершин, побудована за заданою функцією апроксимації координат, які лежать між основними вершинами. За допомогою сплайнів і модифікатора **Lathe** зручно отримувати симетричні об’єкти, які мають вісь обертання (рис. 9.3).

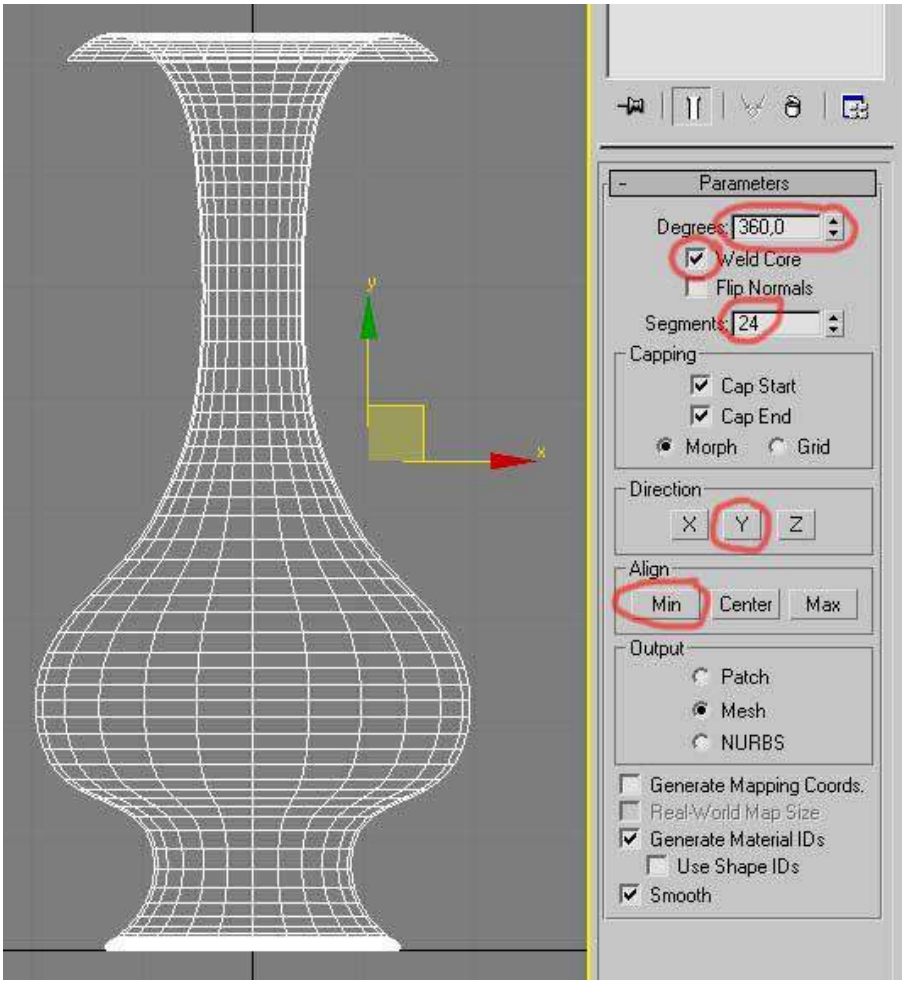


Рисунок 9.3 – Об’єкт з віссю обертання

При моделюванні за допомогою сплайнів основу конструкції тривимірних об’єктів складають відрізки прямих і кривих ліній, званих

сплайнами. Об’єкт-сплайн складається з вершин (серед яких одна позначається білим квадратиком і вважається першою) і сегментів (кожен з них незалежно від його кривизни складається з більш дрібних прямолінійних відрізків – кроків). Сплайн, що не має розривів по периметру, називається замкнутою формою.

Кожна вершина сплайна має дотичні вектори, забезпечені на кінцях маркерами, які керують кривизною сегментів сплайна при вході у вершину і виході з неї. Залежно від властивостей дотичних векторів розрізняють типи вершин: згладжена, зі зламом, Безьє, Безьє зі зламом. Спеціальний тип сплайнів – неоднорідні раціональні **В-сплайни (НРВС)** (рис.9.4), які можуть бути створені на основі контрольних точок (крива проходить через них) або на основі керуючих вершин (виконують роль вузлів решітки деформації).

Побудова гладких кривих або поверхонь відповідно до набору заданих вершин є типовим завданням комп’ютерної графіки.

У тому випадку, якщо ми використовуємо кубічні В-сплайни, з заданої послідовності вершин необхідно вибрати дві сусідні вершини, після чого між ними буде побудована крива кубічного полінома на основі позиції чотирьох точок: дві вибрані і дві сусідні з ними.

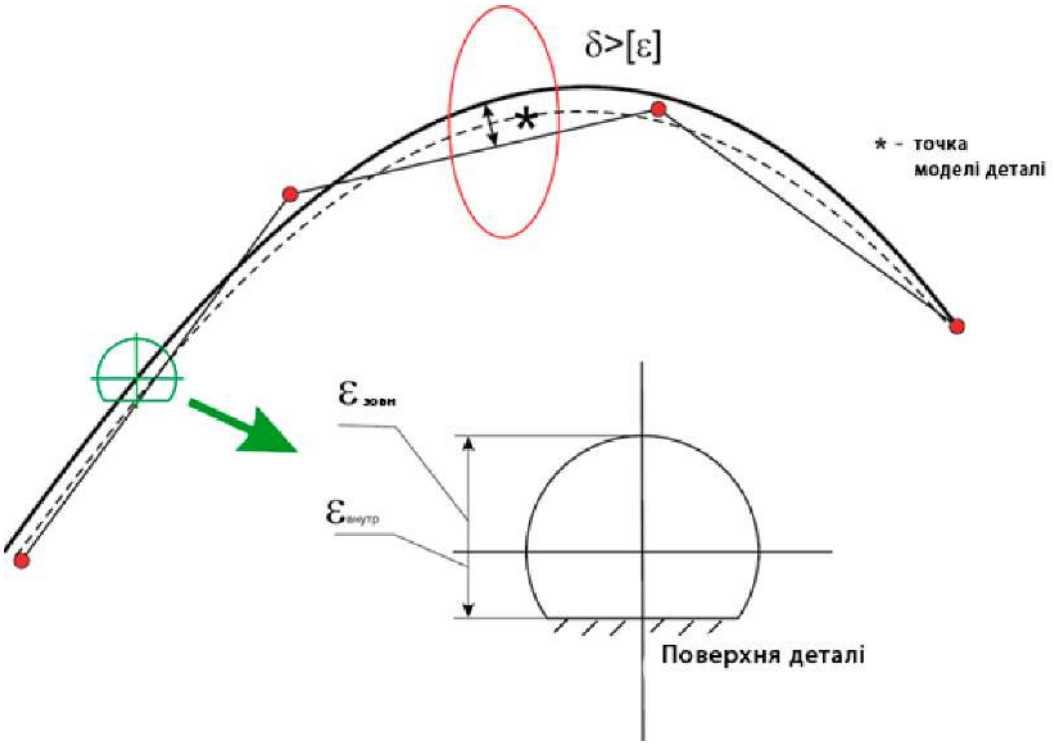


Рисунок 9.4 – В-сплайн

Таким чином, **B-сплайн** дозволяє отримати більш згладжені криві за рахунок того, що створювана крива не пройде точно через задані точки.

3D Studio Max підтримує цілий набір команд створення сплайнів різних типів, найпростішою з яких є команда **Line** (Лінія). Ця команда дозволяє створювати лінії практично будь-якої необхідної форми і є найбільш гнучкою з усіх команд створення типових сплайнів. До інших типів відносять кола, дуги, символи тексту і ще ряд об'єктів. Створюючи один або кілька сплайнів і комбінуючи їх між собою, можна побудувати безліч різних форм, необхідних для моделювання сцени.

Видавлений сплайн – це сплайн, якому надається товщина в певному напрямку. У багатьох випадках цей метод виявляється простішим, ніж застосування тривимірних примітивів для отримання того ж результату. Можна уявляти собі процес видавлювання і як створення копії сплайна, що поміщається точно над оригіналом і на деякій відстані від нього, в результаті чого утворюються верхня і нижня основи об'єкта з подальшою побудовою бічної поверхні по периметру основи.

Застосування методу видавлювання при моделюванні виявляється дуже зручним: можна видавити рядок тексту, щоб отримати об'ємний текст; різноманітні плоскі форми можуть служити заготовками для формування методом видавлювання таких об'єктів, як стіни, компакт диски, форми для печива і т.ін.

Метод видавлювання особливо підходить для об'єктів, що мають один характерний профіль у всіх перетинах за висотою. Операція видавлювання здійснюється завдяки застосуванню до виділеної форми модифікатора **Extrude** (видавлювання) для отримання об'єкта доданням сплайну висоти, або **Bevel** (скіс) для видавлювання зі скосом бічних граней; після чого на командній панелі **Modify** можна змінити висоту об'єкта або вісь, уздовж якої виконується видавлювання (за замовчуванням це вісь *Z*).

Точка **Прив'язки** – (**Anchor Point**) – точка, або вісь, синхронізуюча координати переміщення вершин, або полігонів, у тривимірному просторі. Також служить для групування об'єктів і прискорення роботи над моделюванням об'єктів. Точкою прив'язки також може бути пустий (неіснуючий **Null**) об'єкт.

NURBS (**Non-Uniform Rational B-Splines**) – перекладається як «неоднорідний раціональний B-сплайн». Це особлива технологія, призначена для створення плавних органічних форм і моделей, базована на складному

математичному апараті. Всього існує близько 1500 рівнянь для опису всіх геометричних елементів, від найпростіших кривих до складних поверхонь.

Через особливості будови **NURBS** поверхні завжди гладкі (у них немає гострих країв, властивих полігонам), тому вони широко використовуються в органічному моделюванні (подібному створенню рослинних форм), для створення моделей тварин, людей, машин і т. д. **NURBS** поверхні не складаються з сітки прямокутників, розбиття поверхонь на багатокутники відбувається лише на етапі рендеринга і передбачає використання оптимального алгоритму для збереження гладкості. Тому при будь-якому наближенні дотримується гладкість поверхні.

Існує два типи **NURBS** кривих і поверхонь: **Point** (рис. 9.5), і **CV (Control Vertex)** (рис. 9.6). Різниця між ними полягає лише в способі управління. Об'єкт **Point** керується точками, що лежать безпосередньо на самому об'єкті, іншими словами, об'єкт проходить через ці точки. Об'єкт **CV** керується вершинами, які розташовуються поза об'єктом і пов'язані між собою лініями. Це нагадує керуючі вершини Безьє, які застосовуються в технології моделювання на основі клаптів. Однак існує важлива відмінність: керуючі точки Безьє впливають відразу на всю поверхню, тоді як керуючі вершини **NURBS** впливають на локальну область, розміром якої можна керувати, використовуючи вагу.

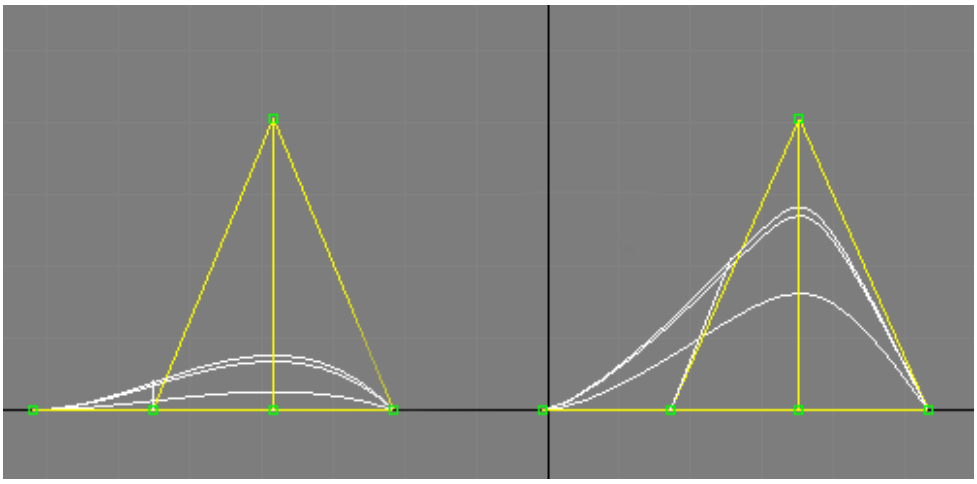


Рисунок 9.5 – Крива сплайн типу **Point**

Одна поверхня або крива не можуть одночасно керуватися як точками, так і вершинами, але всередині одного об'єкта **NURBS** можуть міститися як підоб'єкти **Point**, так і **CV** (рис. 9.6).

Рисунок 9.6 – Крива сплайн типу **CV**

Використання геометричних примітивів типу **NURMS – (Non-uniform rational mesh smooth)**: геометричний примітив, широко використовуваний для опису кривих поверхонь. Щоб зробити будь-який полігональний об’єкт більш округленим і плавним, використовується модифікатор **Mesh Smooth (Згладжування сітки, Згладжування каркаса). Модифікатор Mesh Smooth** дозволяє згладжувати об’єкти за технологією **NURMS**, тобто шляхом збільшення кількості полігонів, клонуванням з поточних полігонів, вершин і граней. Модифікатор «**Mesh smooth**» в 3DS MAX оперує застосуванням даного алгоритму до низькополігонального каркаса, створюючи високополігональну згладжену поверхню. Крім того, один з фільтрів пакета Adobe Illustrator використовує той самий алгоритм згладжування, але тільки для двомірних кривих**.**

Об’єкти – власне графічні об’єкти-примітиви типу Shape.

Субоб’єкти – об’єкти, що входять до складу об’єкта типу Shape. А саме це вершини, ребра, полігони, на яких можуть бути застосовані модифікатори.

Шейдер – (англ. **Shader**; схема затемнення, програма побудови тіней). Це програма обробки (розрахунку) полігонів у тривимірній графіці для визначення остаточних параметрів візуалізації об’єкта або зображення.

Вона може включати в себе опис поглинання і розсіяння світла довільної складності, накладення текстури, віддзеркалення і заломлення, затемнення, зміщення поверхні і ефекти пост-обробки.

Програмовані шейдери гнучкі і ефективні. Складні на вигляд поверхні можуть бути візуалізовані за допомогою простих геометричних форм. Наприклад, шейдери можуть бути використані для малювання поверхні з тривимірної керамічної плитки на абсолютно плоскій поверхні.

9.2.1. Вершинні шейдери (Vertex Shader)

Вершинний шейдер оперує даними, зіставленими з вершинами багатогранників. До таких даних, зокрема, належать координати вершини у просторі, текстурні координати, тангенс-вектор, вектор бінормалі, вектор нормалі. Вершинний рейдер (рис. 9.7) може бути використаний для видового і перспективного перетворення вершин, генерації текстурних координат, розрахунку освітлення і т. д.

Рисунок 9.7 – Вершинний шейдер

9.2.2. Геометричні шейдери (Geometry Shader)

Геометричний шейдер (рис. 9.8), на відміну від вершинного, здатний обробити не тільки одну вершину, а й цілий примітив. Це може бути відрізок (дві вершини) і трикутник (три вершини), а при наявності інформації про суміжні вершини (adjacency) може бути оброблено до шести вершин для трикутного примітиву. Крім того, геометричний шейдер здатний генерувати

примітиви «на льоту», не задіюючи при цьому центральний процесор. Вперше почав використовуватися на відкритих Nvidia серії 8.

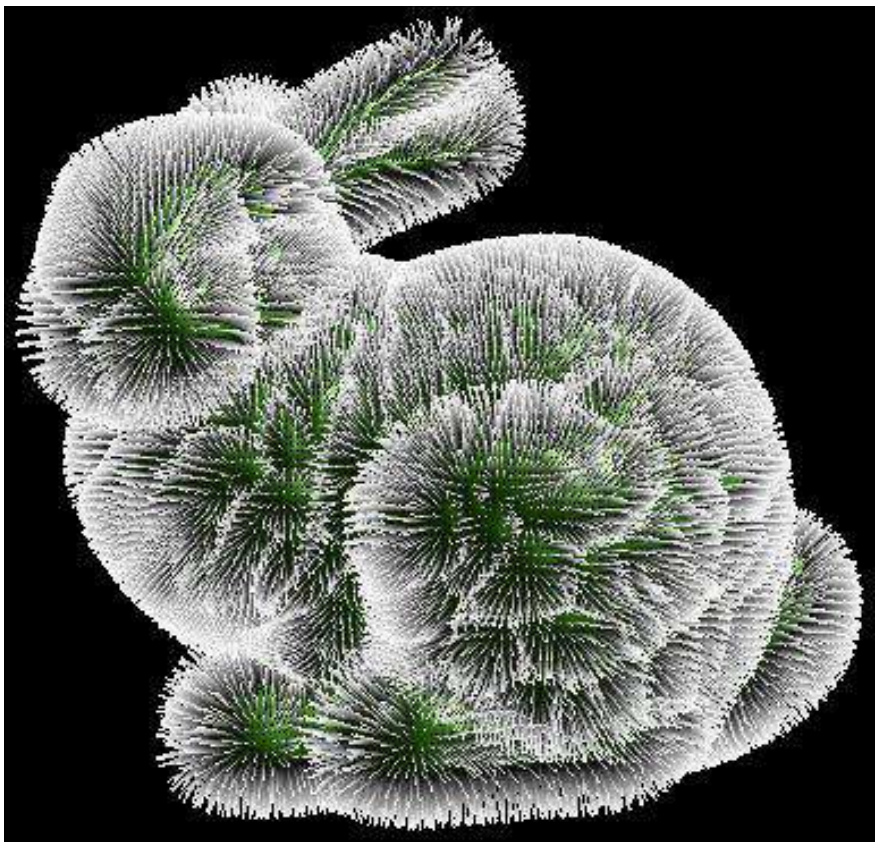


Рисунок 9.8 – Приклад покриття об’єкта текстурою, створеної за допомогою геометричного шейдера

9.2.3. Піксельні шейдери (Pixel Shader)

Піксельний шейдер працює з фрагментами растрового зображення (рис. 9.9). Під фрагментом зображення в даному випадку розуміється піксель, якому поставлений у відповідність певний набір атрибутів, таких як колір, глибина, текстурні координати. Піксельний шейдер використовується на останній стадії графічного конвеєра для формування фрагмента зображення.

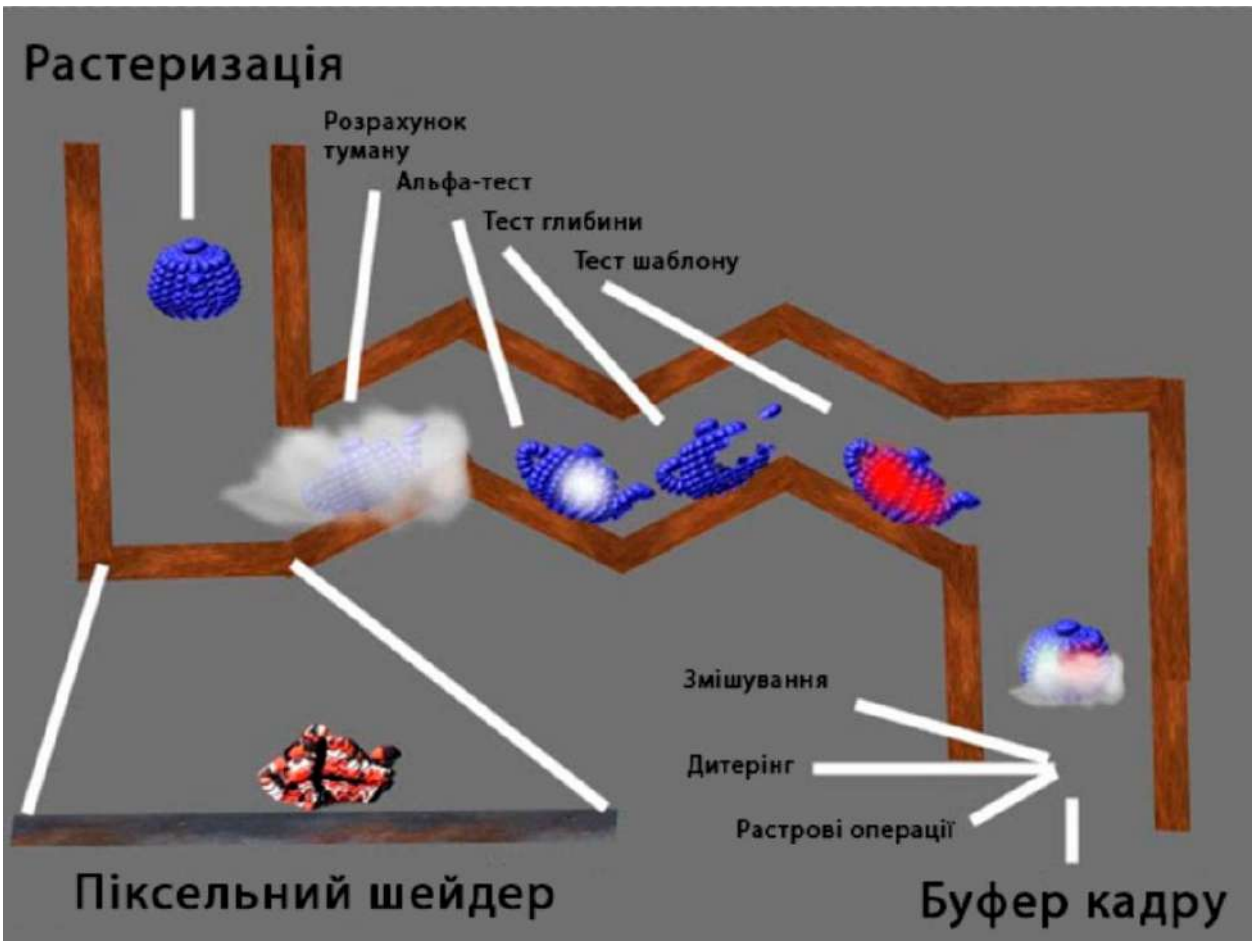


Рисунок 9.9 – Піксельний шейдер

9.3. Лофтінг

Лофтінг (lofting) – це спосіб створення об’єктів з плоских форм шляхом формування оболонки за опорними перетинами, що розставляють уздовж заданої траєкторії довільної форми. Оболонка заповнює відстані між перетинами уздовж зазначеного шляху, в результаті чого виходить тривимірна модель. За допомогою лофтінга створюються тривимірні об’єкти, коли опорні перерізи форм поміщаються уздовж деякої направляючої. У міру того як в набір додаються нові форми, на них будується лофтінгова поверхня, або оболонка, що підлаштовується під контури кожної форми. Лофтінговий шлях може бути будь-яким, але це обов’язково повинен бути один безперервний сплайн (рис. 9.10).

Даний метод моделювання прекрасно підходить для тих моделей, форма яких може бути охарактеризована певним набором поперечних перерізів.

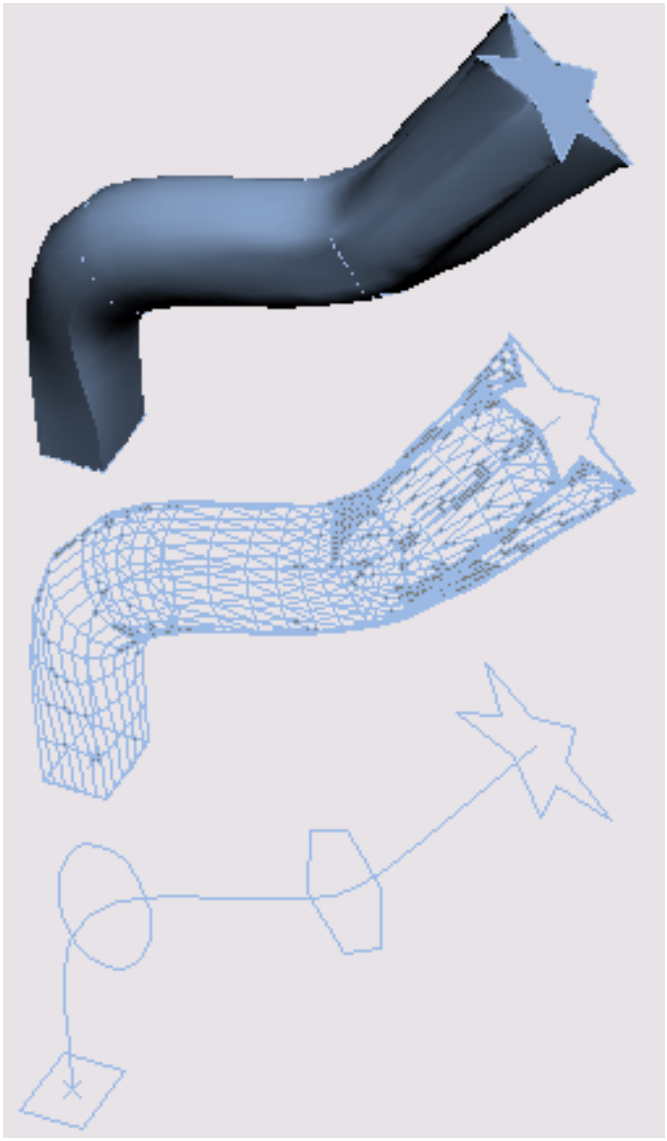


Рисунок 9.10 – Процес створення лофтінгового об’єкта

Підоб’єкти, складові лофтінгового об’єкта, – це шлях лофтінга і форми опорних перетинів (рис. 9.11). Можна редагувати лофтінгові об’єкти, здійснюючи правку форм і шляхів на рівні підоб’єктів або ж працюючи з екземплярами форм, з яких був побудований лофтінговий об’єкт (рис. 9.11).

Робота з оригіналами форм – найпростіший спосіб редагування. Потрібно лише тільки виділити форму, відкрити панель **Modify** і змінити або налаштувати параметри створення форми. Ніякі перетворення оригінальних форм не діють на побудований з них лофтінговий об’єкт до тих пір, поки ви не застосуєте до них модифікатор **XForm**.

У режимі редагування підоб’єктів можна переміщати і обертати форми опорних перетинів і змінювати їх масштаб. Можна також обертати шлях лофтінга щодо локальної осі **Z**. Оскільки виділення підоб’єктів лофтінга не

заноситься в стек модифікаторів, будь-яке застосування модифікаторів на рівні підоб'єктів може призвести до несподіваних результатів або до зміни всього лофтінгового об'єкта.

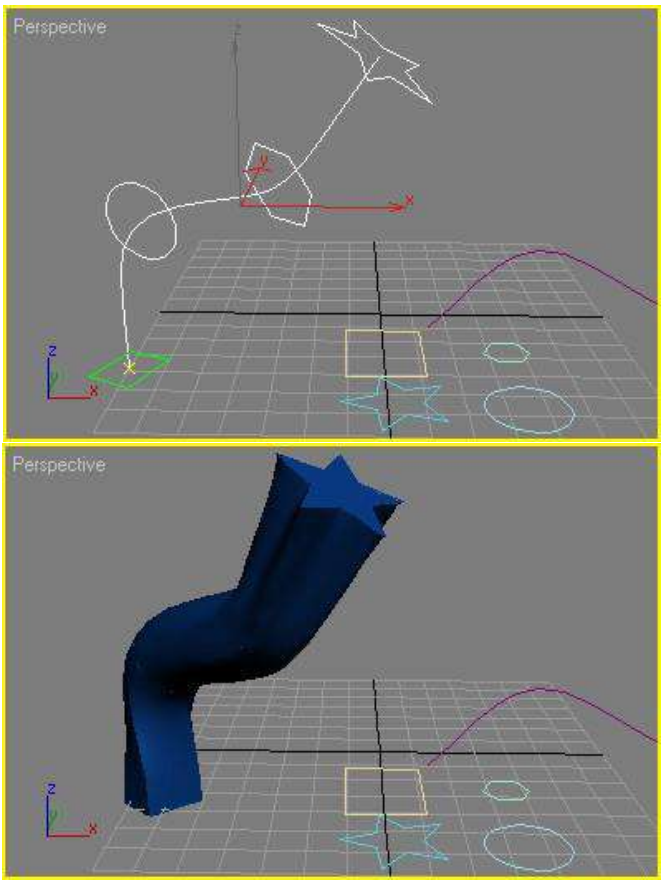


Рисунок 9.11 – Підоб'єкти лофтінгового об'єкта

Анімувати лофтінгові об'єкти можна на рівнях об'єктів або підоб'єктів. З цілим лофтінговим об'єктом анімація здійснюється так само, як і з будь-яким іншим сітчастим об'єктом. На рівні підоб'єктів можна здійснити анімацію оригіналів лофтінгових елементів (тобто зразків, з яких підоб'єкти були отримані методом Instance), а також шляхів і опорних перетинів лофтінгового об'єкта. Застосування перетворень або модифікаторів безпосередньо до лофтінгових підоб'єктів не веде до генерування ключів анімації, при цьому лофтінговий об'єкт не змінюється з плином часу.

9.3.1. Морфінгові об'єкти

Тривимірний **морфінг (morphing)** – це метод анімації, коли деякий об'єкт, званий початковий (seed object), змінює свою форму відповідно до того, як виглядають об'єкти-цілі (target). Цей метод часто використовується в

анімації персонажів для зміни виразу обличчя і при імітації руху губ при розмові (рис. 9.12).

Морфінг може бути застосований до сіткових об'єктів, патчей або NURBS-поверхонь. *Надалі ми розглянемо* ***сітковий морфінг***. Однак принципи його єдині для об'єктів будь-якого походження. Необхідна умова створення морфінгом об'єктів – збереження числа вершин в цілях: воно повинно бути таким же, як в початковому об'єкті. Це пояснюється тим, що операція морфінга просто переміщує вершини початкового об'єкта так, щоб вони збігались з відповідними вершинами об'єкта-цілі. Якщо число вершин змінюється, морфінгова анімація не працює.

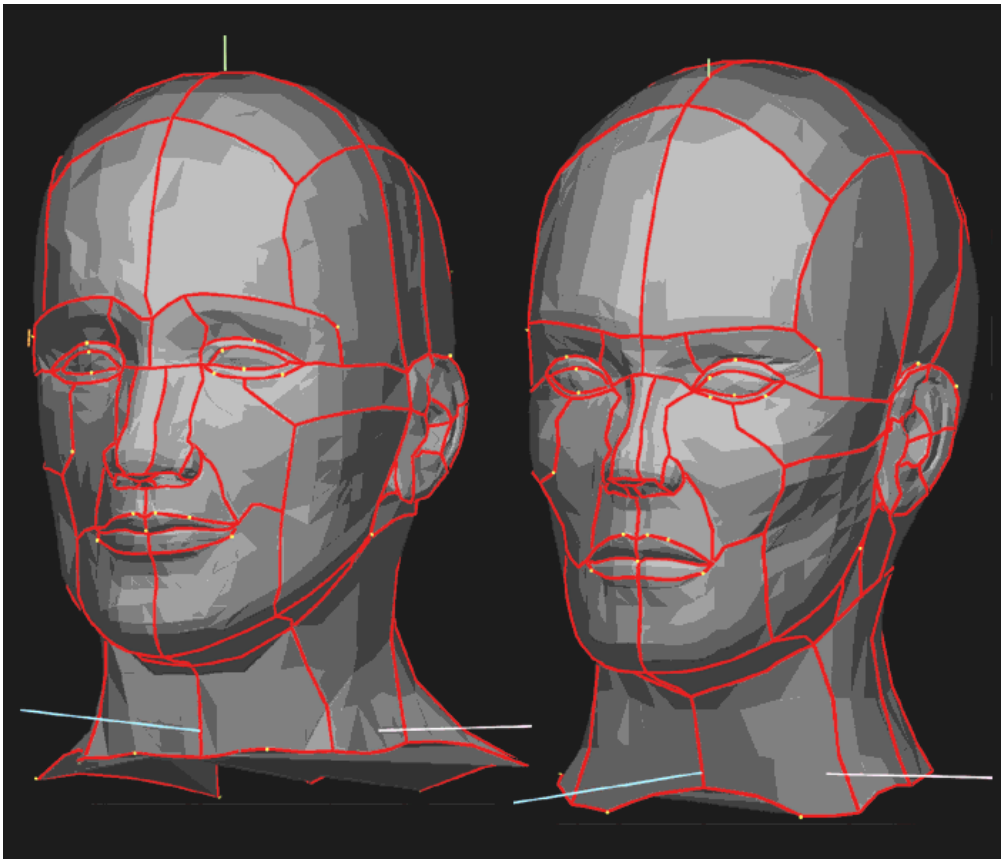


Рисунок 9.12 – Створення другого об'єкта з першого за допомогою морфінга

Разом з модифікатором Morpher застосовується особливий тип матеріалу – ***Morph****, який гнадає можливість призначати різні матеріали для каналів модифікатора, дозволяючи робити морфінг матеріалів.*

9.3.2. Створення узгодженого об'єкта

Узгоджені об'єкти (conform objects) – це складові об'єкти, створені шляхом «обгортання» вершин одного об'єкта навколо вершин іншого.

Поверхня першого об’єкта, званого об’єктом обгортки (wrapper object), повинна узгоджуватися з поверхнею другого, який називається той, що обгортається (wrap-to object). Розміщуючи об’єкт, що обгортає, навколо об’єкта, який обгортаєте, ви приблизно повторюєте форму об’єкта, що обгортаєте, як наче б ви створювали на ньому тонкий наліт.

9.3.3. З’єднання об’єктів

Команда **Connect (З’єднати)** з’єднує два або більше сітчастих об’єктів, вибудовуючи мости між отворами в їх поверхнях (рис. 9.13). Використовуйте цю команду для створення архітектурних структур, меблів, пристосувань, інструментів та інших предметів штучного походження. Команду **Connect** можна також використовувати для приєднання пальців до кисті або кінцівок до тулуба.

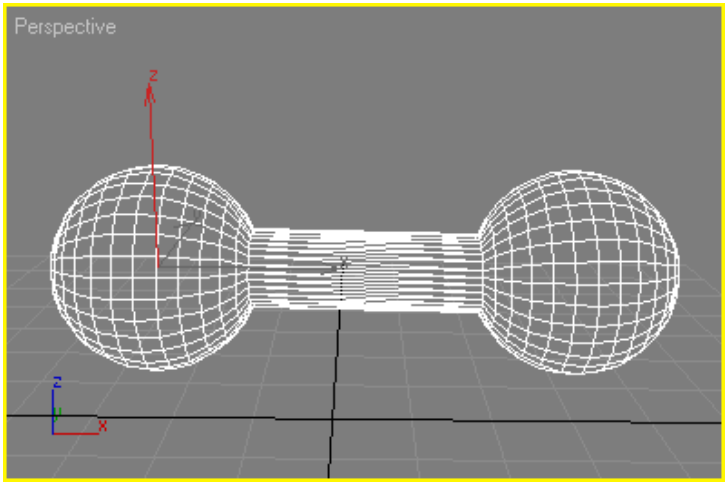


Рисунок 9.13 – З’єднання двох об’єктів командою **Connect**

Операція **Scatter (Розподілити)** розподіляє клони вихідного об’єкта по поверхні або обсягу об’єкта розподілу.

9.3.4. Створення ландшафту

Ландшафти (terrains) – це тривимірні об’єкти, які можна створювати з контурних даних. Ландшафти використовуються для побудови сайтів, вивчення тіней, складання планів і при створенні ігор. Якщо у вас немає підхожої контурної моделі, ви можете створити її, використовуючи масив близько розташованих сплайнів зі злегка зменшуючимися від рівня до рівня розмірами (рис. 9.14).

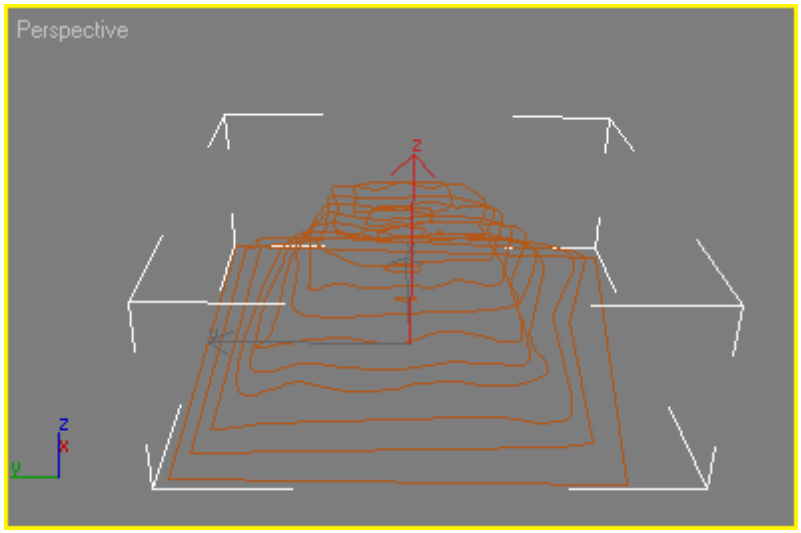


Рисунок 9.14 – Приклад Ландшафту в 3DS MAX

9.4. Булеві операції

Одним з найбільш зручних і швидких способів моделювання є створення тривимірних об’єктів за допомогою булевих операцій.

Наприклад, якщо два об’єкти перетинаються, на їх основі можна створити третій об’єкт, який буде результатом додавання, віднімання або перетину вихідних об’єктів.

У 3DS MAX є два незалежних один від одного набору інструментів для роботи з булевими операціями – складений об’єкт **(Compound Objects)**, **Boolean (Булева операція)** і модуль **Pro Booleans**. У більшості випадків має сенс використовувати саме модуль **Pro Booleans**, оскільки результати його роботи більш коректні. Наприклад, маємо два об’єкти – циліндр і кулю.

При використанні складеного об’єкта **Boolean (Булева операція)** необхідно виконати такі дії:

– виділити перший об’єкт (об’єкт А), який братиме участь в утворенні моделі, створеної після виконання булевої операції;

– перейти на вкладку **Create (Створення)** в командній панелі, вибрати в категорії **Geometry (Геометрія)** рядок **Compound Objects (Складові об’єкти)** і натиснути кнопку **Boolean (Булева операція)**;

– встановити параметри булевої операції;

– натиснути кнопку **Pick Operand B (Вибрати операнд)** у згортку **Pick Boolean (Вибрати логічний об’єкт)** і клацнути на другому об’єкті (об’єкт В, який братиме участь в операції).

При використанні модуля **Pro Booleans** порядок дій залишається тим же, однак замість складеного об’єкта **Boolean (Булева операція)** використовується складений об’єкт **ProBoolean (Пробулеві об’єкти)**. Параметри булевої операції вказуються в налаштуваннях цього об’єкта, а для початку виконання обраної операції потрібно натиснути кнопку **Start Picking (Почати вибір)**.

Існує чотири типи булевих операцій: **Union**, **Intersection**, **Subtraction**, **Cut**.

9.4.1. Union (Додавання)

Булеве складання об’єктів передбачає побудову моделі на основі поверхонь двох і більше об’єктів. При використанні булева складання об’єкти, які беруть участь в операції, стають одним цілим, тобто на їх основі формується єдиний об’єкт (рис. 9.15).

Зовні поверхня, отримана в результаті булевого складання, і поверхня згрупованих об’єктів здаються однаковими, проте між ними є істотні відмінності. По-перше, при виконанні булевого складання відсікаються невидимі ділянки об’єктів. По-друге, топологія ребер і вершин отриманої поверхні відрізняється від полігональної структури вихідних об’єктів.

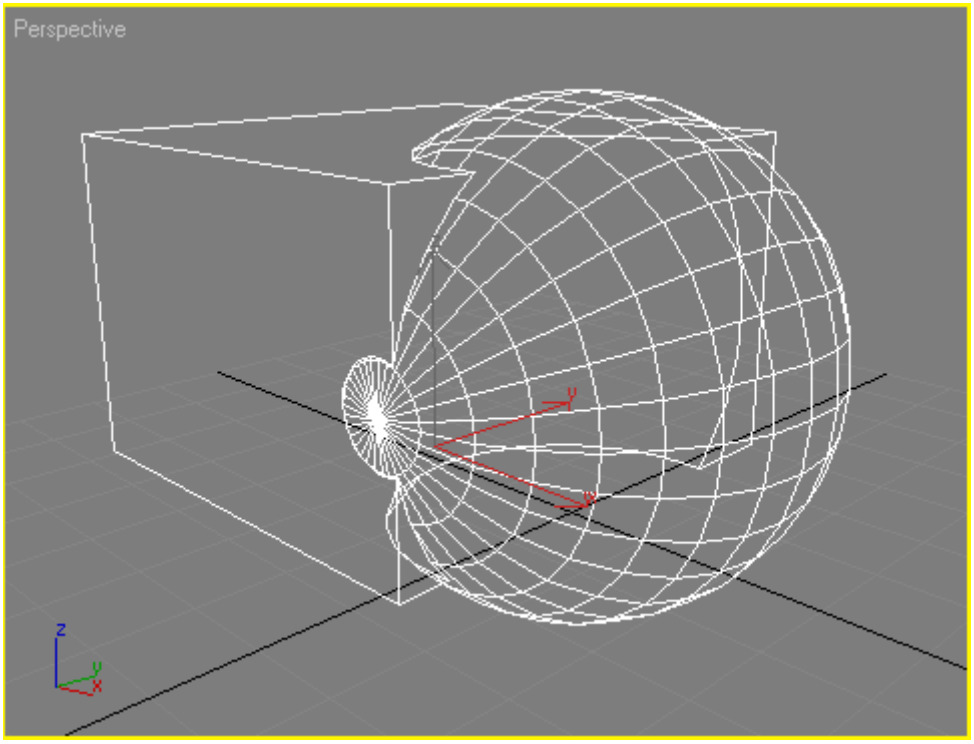


Рисунок 9.15 – Результат виконання операції **Union**

Якщо об’єкти відображаються в режимі **Smooth + Highlights (Згладжування і відблиски)** і при цьому допоміжний режим **Edged Faces (Контури ребер)** відключений, то визначити, що операція булевого складання виконана, ви зможете за зміною кольорів. В цьому випадку всі об’єкти, які брали участь в операції, змінять свій колір на колір першого об’єкта. Це буде вказувати на те, що на їх основі утворена єдина поверхня.

Спостерігати дію операції **Union (Складання)** найзручніше в режимі відображення **Wireframe (Каркас)**, в якому видно сітчасту оболонку об’єкта.

9.4.2. Intersection (Перетин)

Булевий перетин означає відсікання всіх непересічних частин об’єктів, які задіяни в операції. Іншими словами, утворений в результаті виконання цієї операції об’єкт буде мати форму, спільну для пересічних поверхонь. Оскільки область перетину циліндрів невелика, результатом виконання операції перетину буде невеликий об’єкт (рис. 9.16).

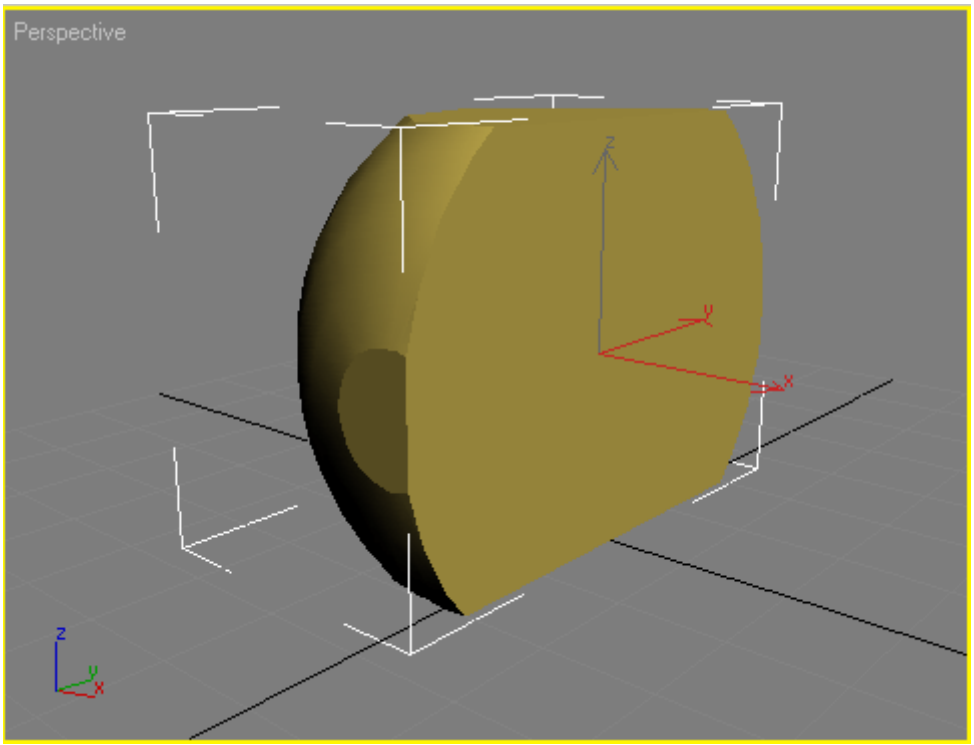


Рисунок 9.16 – Результат виконання операції **Intersection**

9.4.3. Subtraction (Віднімання)

Булеве віднімання – це операція, протилежна булевому перетину. В результаті її застосування буде утворена модель, яка включає в себе ту частину першого об’єкта, задіяного в операції, що не перетинається з другим об’єктом.

При виконанні цієї операції складеним об’єктом **Boolean (Булева операція)** можна вказати, який об’єкт з якого вираховується: перший з другого (**Subtraction** (B-A)) або другий з першого (**Subtraction** (A-B)).

Припустимо, нам потрібно вирізати в циліндрі той обсяг, який займає в ньому куля. Для цього відзначаєте циліндр і вибираєте **Create - Geometry - Compound Objects - Boolean** (рис. 9.17).

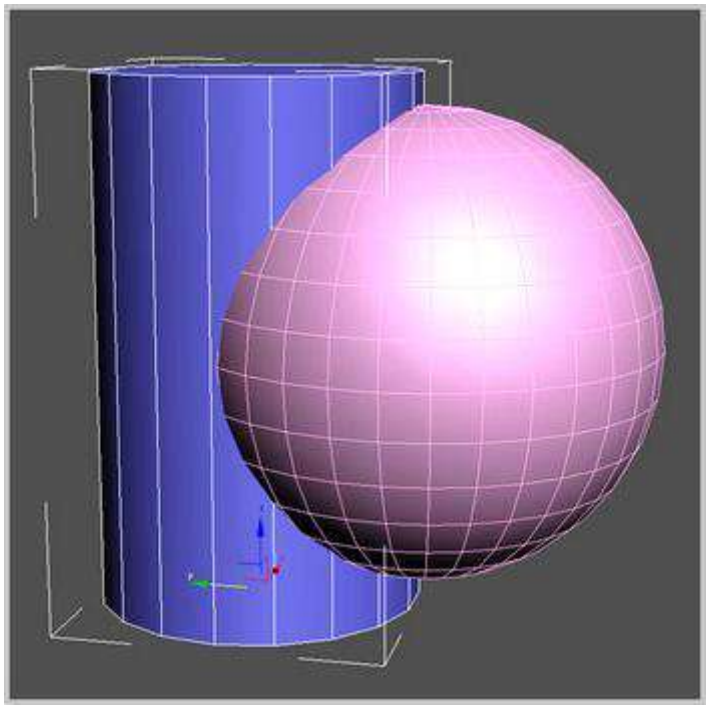


Рисунок 9.17 – Результат виконання операції **Subtraction**

В параметрах натискаєте на **Pick Object** і вказуєте на кулю, в результаті отримуєте необхідне.

Кількість сегментів цього вирізу буде точно такою ж, як була на кулі.

При роботі з складеним об’єктом **ProBoolean** (Пробулеві об’єкти) не можна вказати порядок виконання операції. З того об’єкта, який був виділений першим, буде виконано віднімання другого об’єкта.

9.4.4. Cut (Розрізування)

Ця операція призначена для розрізання одного об’єкта іншим. Лінія розрізу проходить по тому місцю, де два об’єкти перетинаються, і її форма визначається формою другого об’єкта, що задіяни в операції. При виконанні цієї операції за допомогою складеного об’єкта **Boolean (Булева операція)** можна вибрати один з чотирьох типів даної операції, які застосовуються в різних випадках.

При використанні варіанта **Refine (Деталізація)** в топологію результуючого об’єкта включаються додаткові ребра по периметру перетину оболонок об’єктів (рис. 9.18). Щоб розбити об’єкт на два елементи, має сенс вибирати варіант **Split (Розділити)**.

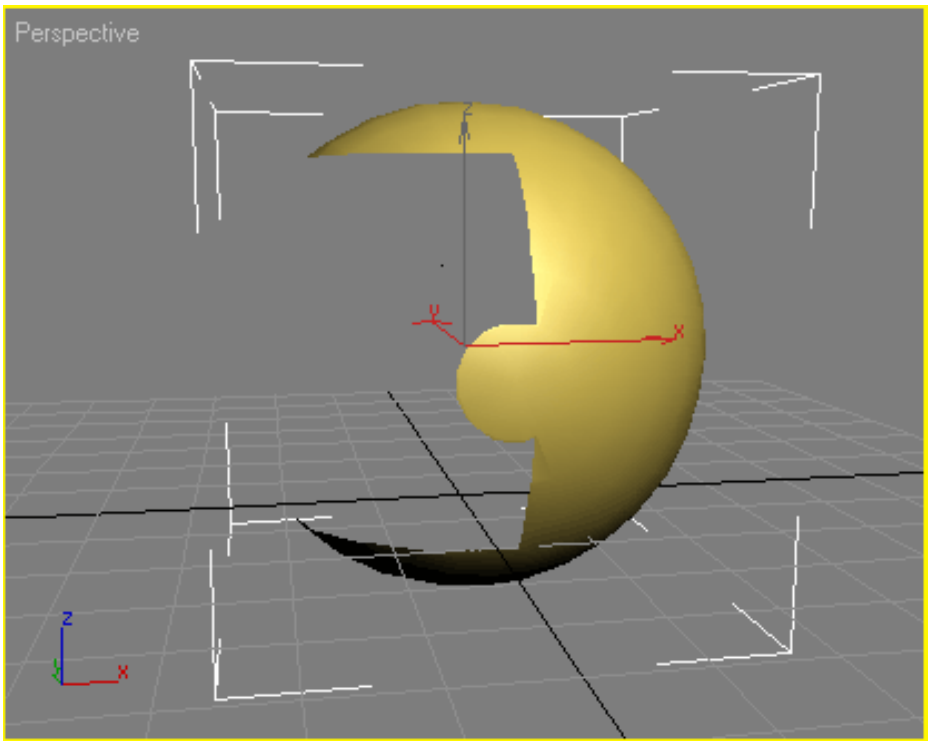


Рисунок 9.18 – Результат виконання операції **Cut**

Примітка. *Працювати з кожним з елементів ви зможете після перетворення об’єкта в редаговану поверхню або призначення йому відповідного модифікатора.*

Результати, отримані при виборі варіантів **Remove Inside (Видалити всередині)** (рис. 9.19) і **Remove Outside (Видалити зовні)** (рис. 9.20), нагадують два варіанти виконання операції **Subtraction (Віднімання)**. Різниця полягає в тому, що в даному випадку в тому місці, де поверхні взаємодіючих об’єктів перетинаються, утворюється отвір.

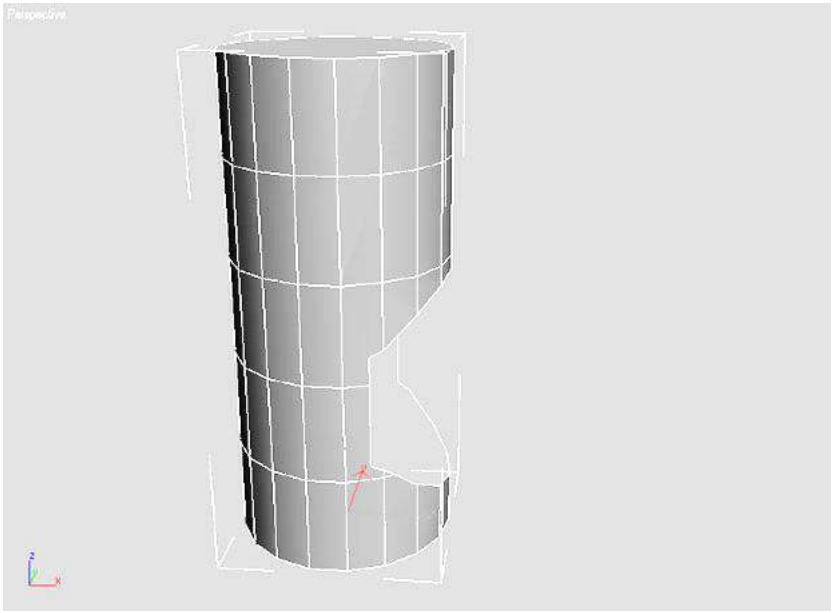


Рисунок 9.19 – Результат виконання операції **Cut (Розрізування)** з обраним варіантом **Remove Inside (Видалити всередині)**

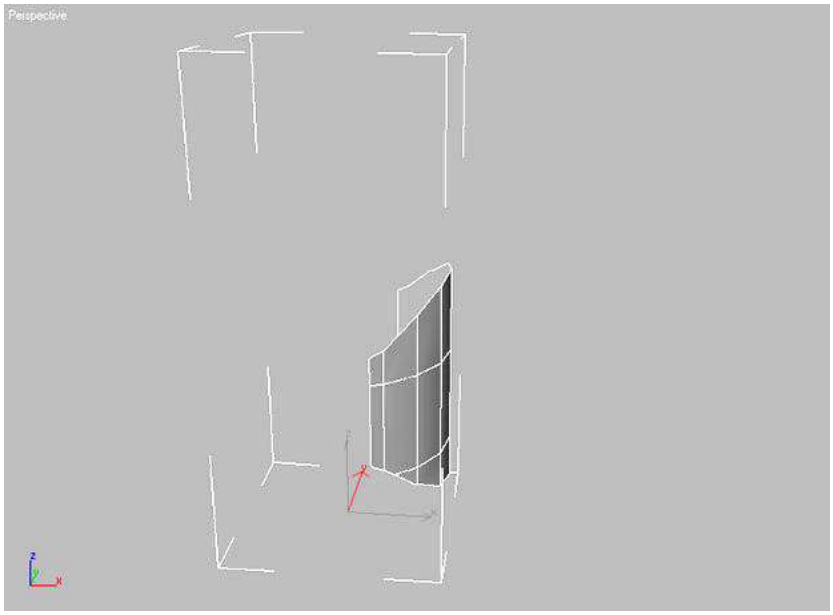


Рисунок 9.20 – Результат виконання операції **Cut (Розрізування)** з обраним варіантом **Remove Outside (Видалити зовні)**

У складеному об’єкті **ProBoolean (Пробулеві об’єкти)** операцію **Cut (Розрізування)** замінюють параметри **Imprint (Відбиток)** і **Cookie (Печиво)**. Вони є взаємовиключними: якщо використовується один з них, то другий застосовувати не можна.

При виконанні булевих операцій з установленим прапорцем **Imprint (Відбиток)** у геометрію другого об’єкта будуть включені нові ребра по периметру перетину двох об’єктів. Параметр **Imprint (Відбиток)** дає

можливість отримати той самийрезультат, незалежно від типу булевої операції. Результат операції з установленим прапорцем **Imprint (Відбиток)** аналогічний результату операції **Cut (Розрізування)** з обраним варіантом **Refine (Деталізація)** при використанні складеного об’єкта **Boolean (Булева операція)**.

Параметр **Cookie (Печиво)** названий так невипадково. Він дозволяє отримати отвір в тому місці, де поверхні взаємодіючих об’єктів перетинаються. Іншими словами, створюється враження того, що з поверхні "вигризли" полігони, як ніби відкусили шматок печива. Виконання булевих операцій **Subtraction (Віднімання)** і **Intersection (Перетин)** з установленим прапорцем **Cookie (Печиво)** подібне до використання операції **Cut (Розрізування)** з обраними варіантами **Remove Inside (Видалити всередині)** і **Remove Outside (Видалити зовні)** при використанні складеного об’єкта **Boolean (Булева операція)**. Використовуйте параметр **Cookie (Печиво)** при виконанні операцій **Subtraction (Віднімання)** і **Intersection (Перетин)**, коли вам потрібно виконати складні за формою отвори.

При роботі зі складеним об’єктом **ProBoolean (Пробулеві об’єкти)** не можна зразу вказати порядок виконання операції. З того об’єкта, який був виділений першим, буде виконано віднімання другого об’єкта.

9.4.5. Об’єкти до і після булевої операції

За замовчуванням при виконанні булевої операції вихідний об’єкт B видаляється. Однак якщо ви хочете, щоб він залишався в сцені в тому вигляді, в якому він був присутній до булевої операції, необхідно змінити положення перемикача у згортку **Pick Boolean (Вибрати булевий об’єкт)**. Цей перемикач має чотири положення:

– **Move (Переміщення)** – якщо перемикач встановлений в це положення, то після виконання булевої операції об’єкт B видаляється. За замовчуванням використовується саме це положення.

– **Copy (Незалежна копія об’єкта)** – при виборі цього положення булева операція виконується з копією об’єкта B, а сам об’єкт залишається у сцені (рис. 9.21).

– **Instance (Прив’язка)** – якщо перемикач встановлений в це положення, то булева операція проводиться з залежною копією об’єкта B. При цьому вихідний об’єкт залишається у сцені, а зміна параметрів одного з об’єктів (вихідного або булевого) тягне за собою зміну параметрів іншого.

– **Reference (Підпорядкування)** – при виборі цього положення створюється копія об’єкта B, яка частково залежить від вихідного об’єкта.

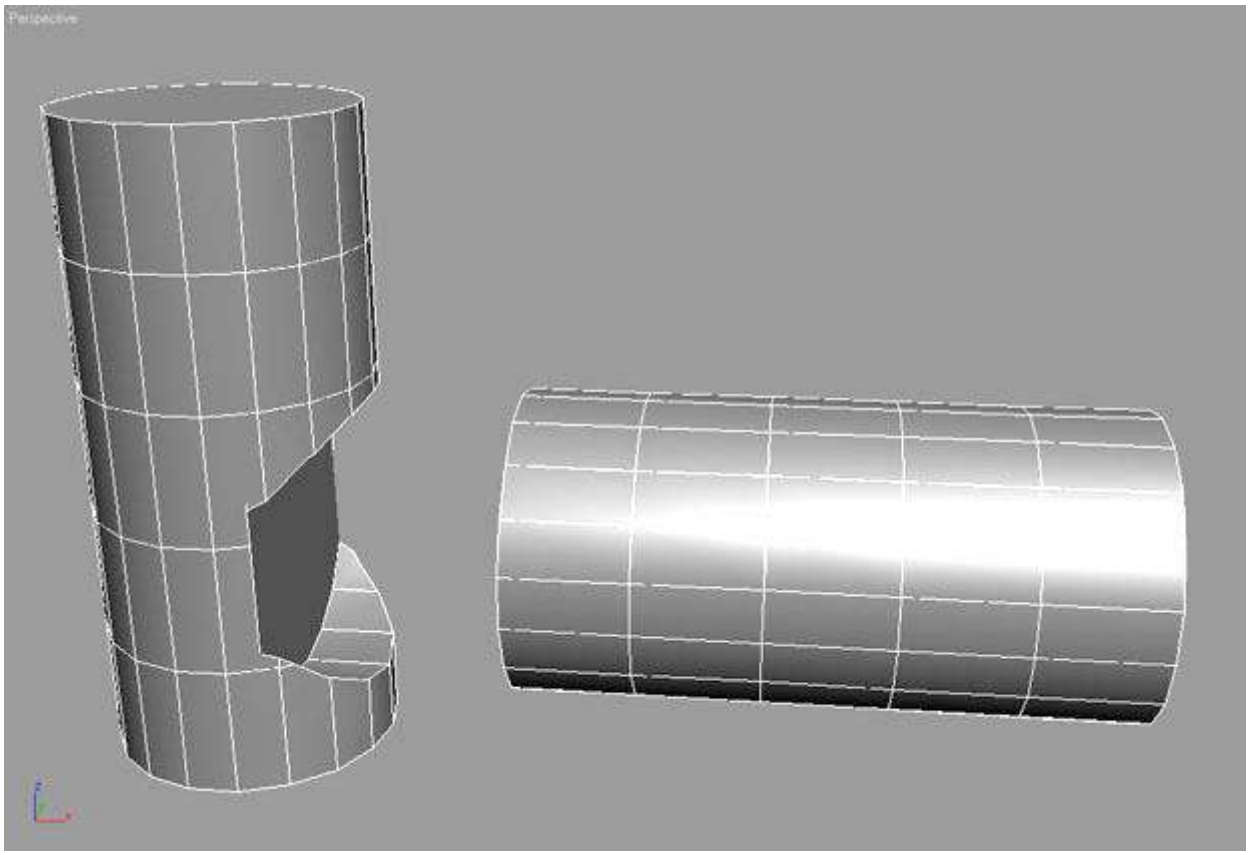


Рисунок 9.21 – Результат виконання булевої операції при положенні перемикача **Copy (Незалежна копія об’єкта)**

Одним з переваг модуля **Pro Booleans** перед об’єктом **Boolean (Булева операція)** є те, що топологія сітки результуючої поверхні може включати в себе чотирикутні грані. Це дає можливість згладити різкі кути на стику булевих об’єктів. Для використання цієї можливості потрібно до виконання операції встановити прапорець **Make Quadrilaterals (Створення чотирикутників)** у свиті **Advanced Options (Додаткові настройки)** параметрів складеного об’єкта **ProBoolean (Пробулеві об’єкти)**. На рис. 9.22 показана топологія об’єкта, отриманого в результаті виконання булевої операції додавання до встановленого прапорцем **Make Quadrilaterals (Створення чотирикутників)**.

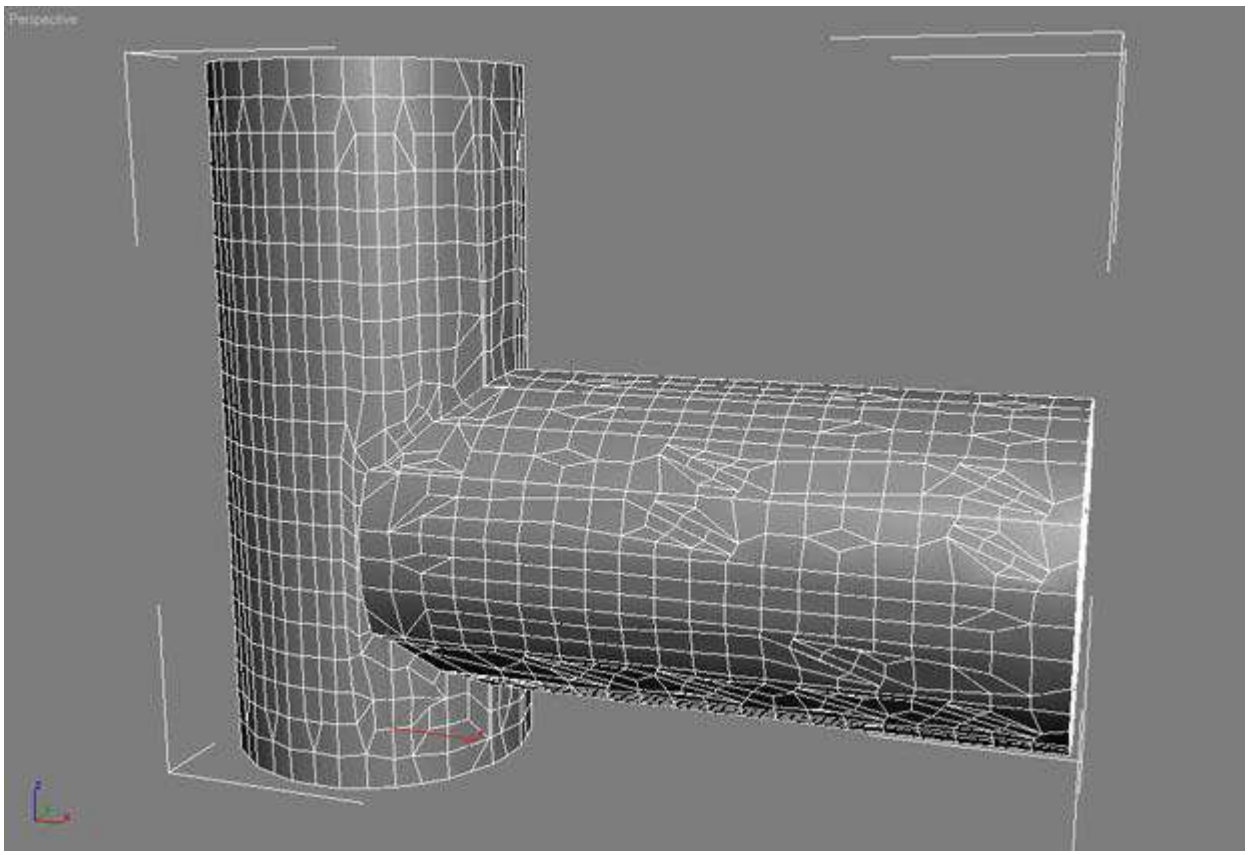


Рисунок 9.22 – Приклад деформації полігональної сітки при неправильному з’єднанні двох об’єктів

Незважаючи на те що булеві операції 3DS MAX 7 широко застосовуються при створенні тривимірних проєктів, вони мають ряд недоліків, і побудова оболонки результуючої моделі нерідко відбувається з помилками. З цієї причини багато розробників тривимірної графіки використовують у своїх проєктах додатковий модуль **Power Booleans**. Він дозволяє створювати моделі з більш точною геометрією, ніж ті, які можна отримати, використовуючи стандартні засоби, а також швидше будує полігональну сітку. Це особливо помітно при роботі з об’єктами, що мають велику кількість полігонів.

9.5. Ієрархія об’єктів у 3DS MAX

Будь-яка сцена складається з величезного числа об’єктів, для зручності роботи з якими використовують різні варіанти їх об’єднання. З одного боку, це дозволяє в подальшому виконувати ряд дій відразу щодо цілої групи об’єктів, а з іншого – призводить до того, що перетворення одного об’єкта автоматично викликає перетворення іншого. Можливі два підходи до формування подібних зв’язків. Можна об’єднати об’єкти в групи, в яких всі об’єкти є підлеглими і рівноправні між собою. А можна встановити між об’єктами взаємини «предок /

нащадок», де предок – старший елемент, а нащадок – підлеглий елемент ієрархічного співвідношення. Вибір варіанта об'єднання об'єктів залежить від конкретної ситуації. Зазвичай в групи об'єднують складові елементи складного об'єкта і часто повторювані набори однотипних об'єктів. А зв'язок «предок / нащадок» встановлюють для тих наборів об'єктів, які при анімації повинні переміщатися узгоджено, але при цьому у дочірніх об'єктів має залишатися можливість здійснення незалежних рухів.

9.5.1. Поняття ієрархії

Ієрархія – це набір об'єктів зі зв'язками між ними, за умови, що об'єкти об'єднані між собою за принципом «предок / нащадок» (рис. 9.23). **Предок (Parent)** – це об'єкт, який контролює поведінку одного або більше нащадків. Деякі предки самі можуть контролюватися іншими предками, мають більш високий рівень ієрархії. **Нащадок (child)** – об'єкт, контрольований предками, який одночасно може виявитися батьківським для інших нащадків, що перебувають за ієрархією нижче нього. Найверхній об'єкт ієрархії (для жодного іншого об'єкта він не є нащадком) керує всіма об'єктами ієрархії і називається кореневим, або **Root**-об'єктом.

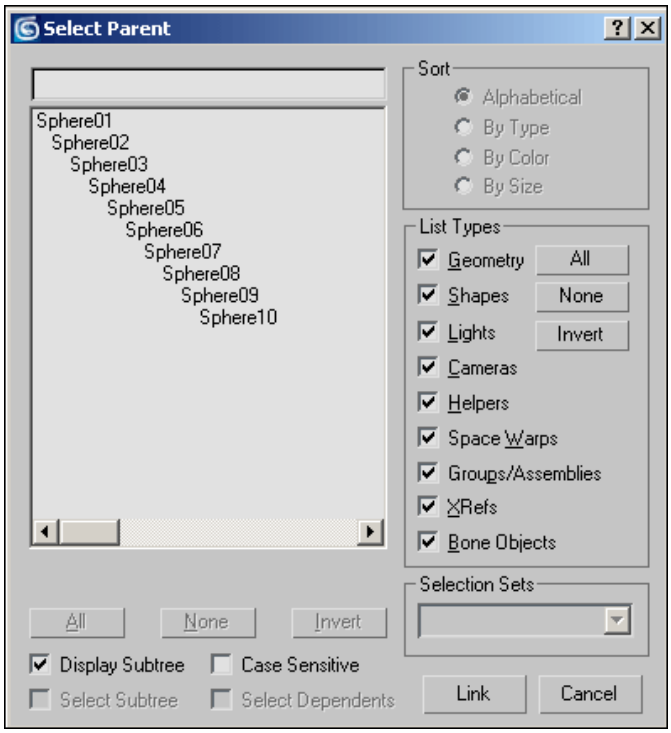


Рисунок 9.23 – Ієрархія в 3DS MAX

Подібний спосіб об'єднання об'єктів дозволяє налаштовувати складні анімаційні руху, при яких шляхом призначення зв'язку «предок / нащадок» можна передавати рух від одного об'єкта до іншого. Класичний приклад

подібної ієрархії – нога людини, що представляє собою ієрархічний ланцюжок з декількох об’єктів: стегна, гомілки, ступні і пальців. Головним в цій структурі є стегно, і його переміщення призводить одночасно до переміщення гомілки, ступні і пальців, причому кожен з даних об’єктів, переміщаючись зі стегном, зберігає можливість рухатися майже незалежно.

Призначення або руйнування зв’язків регулюється розташованими на головній панелі інструментів **SelectandLink (Виділити і зв’язати)** і **UnlinkSelection (Розірвати зв’язок з виділеним)** – рис. 9.24. Перший дозволяє встановити зв’язок між дочірнім та батьківським об’єктами, а другий – зруйнувати зв’язок знову ж між об’єктом-нащадком і об’єктом-предком (при цьому зв’язку з нижчого рівня в ієрархії з дочірніми об’єктами, якщо такі існували, зберігаються).

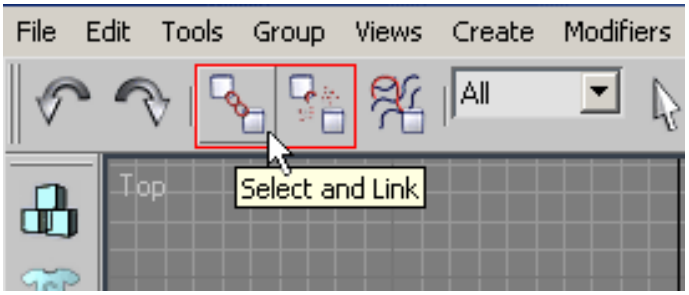


Рисунок 9.24 – Інструмент створення ієрархічного ланцюжка

Переконатися в тому, що потрібні зв’язки встановлені, можна через вікно **Select by Name**.

9.5.1. Керування ієрархічними ланцюжками

Положення і орієнтація пов’язаних в якусь ієрархічну структуру об’єктів не тільки визначається формою ієрархії, але залежить також від обраного методу керування ланцюжком.

Таких методів два:

- пряма кінематика **(Pivot)**;
- інверсна **(IK)** кінематика.

У разі **прямої кінематики (Pivot)** ієрархічні зв’язки передаються від предків до нащадків, тобто дочірні об’єкти успадковують поведінку батьківських, і з ними відбуваються ті ж самі перетворення (переміщення, масштабування і т.д.). У той же час трансформація нащадка не викликає трансформації батьківського об’єкта.

Інверсна кінематика (ІК) відрізняється від прямої іншим принципом успадкування – нащадки можуть надавати рух батьківським об'єктам, тобто програма розраховує положення і орієнтацію батьківських об'єктів, виходячи з положення і орієнтації трансформованого нащадка. Нащадок, який викликає трансформації інших об'єктів за законами інверсної кінематики, називається або ефектором **(Effector)**, якщо він розташований в середині окремої ієрархічної гілки, або кінцевим ефектором **(EndEffector)**, якщо є кінцевим об'єктом даної галузі. Через ефектор здійснюється маніпулювання всього ієрархічного ланцюжка. При цьому трансформація кінцевого ефектора забезпечує трансформацію всіх об'єктів ієрархічної гілки за законами інверсної кінематики. Трансформація просто ефектора призводить до того, що положення об'єктів, що стоять по ієрархії нижче нього, змінюється за законами прямої кінематики, а об'єктів з більш високою ієрархією – за законами інверсної кінематики.

Розглянемо відмінності поведінки пов'язаних об'єктів в кожному з даних методів на прикладі створеного ієрархічного ланцюжка торусом. Виділіть перший в ланцюжку торус **Torus01** і перемістіть його вгору по осі ***Y*** – разом з ним перемістяться і всі інші **Торус**, які є відносно до першого торуса дочірнім. У той же час переміщення останнього торуса ніяк не вплинули на інші, оскільки дочірніх об'єктів у нього немає (рис. 9.25). А тепер увімкніть режим інверсної кінематики (кнопка **Interactive IK**) і повторіть ті ж самі операції з переміщенням першого і останнього **Торус** – переміщення першого торуса дасть ті ж самі результати, що й раніше, а от зміна положення останнього елемента ієрархії призведе до відповідних змін положення інших елементів (рис. 9.26).

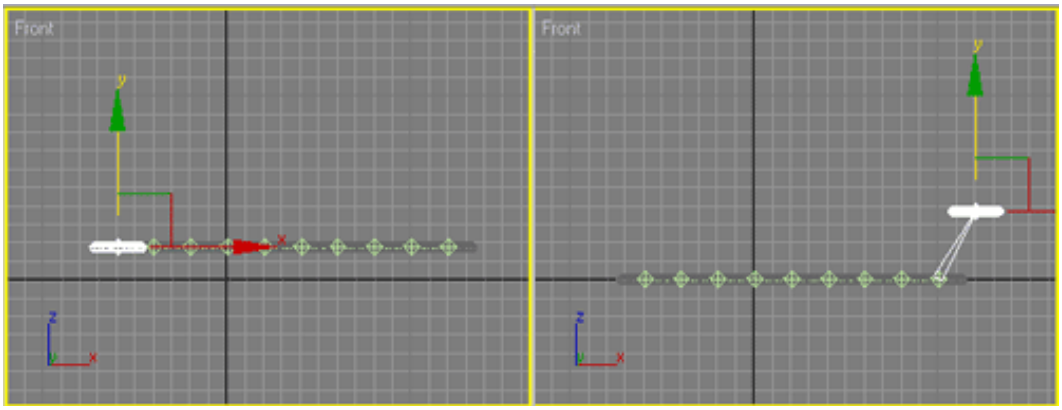


Рисунок 9.25 – Поведінка об'єктів у разі прямої кінематики – при переміщенні першого (зліва) і останнього (праворуч) торусів

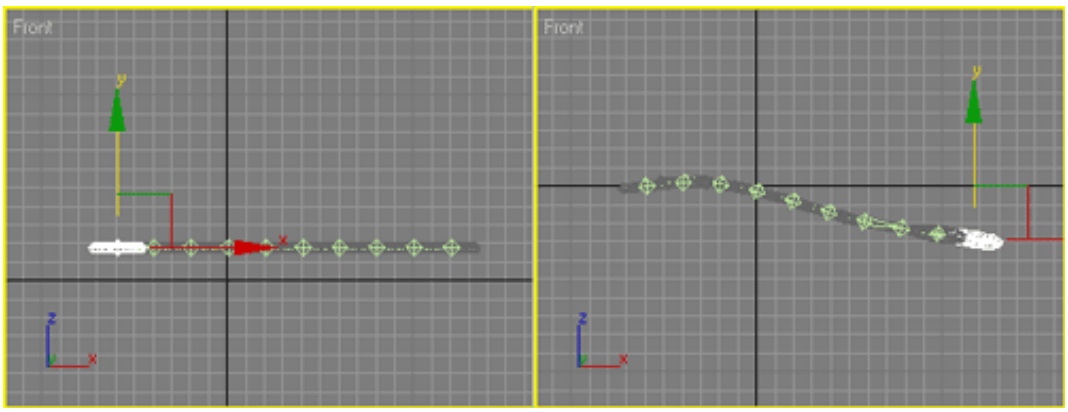


Рисунок 9.26 – Поведінка об’єктів у разі зворотної кінематики – при переміщенні першого (зліва) і останнього (праворуч) торусів

Вибір методу проводиться на панелі **Hierarchy** (рис. 9.27).

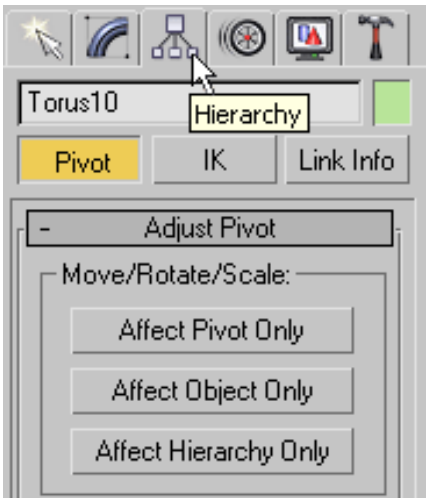


Рисунок 9.27 – Панель **Hierarchy**

Як правило, пов’язані об’єкти повинні трансформуватися тільки якимось певним чином, тобто всі інші варіанти трансформацій для них заборонені. Установлення подібних заборон регулюється через закладку **LinkInfo. Inherit – успадкування** (рис. 9.28).

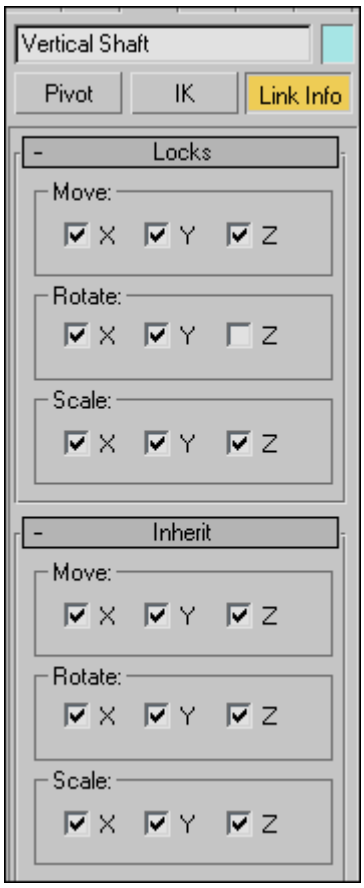


Рисунок 9.28 – Методи спадщини ієрархії

9.5.2. Об'єкти типу Dummy

Нерідко складові елементи ієрархічного ланцюжка поряд зі звичайними, що візуалізуються об'єктами використовуються фіктивні об'єкти-пустушки – об'єкти типу **Dummy** (рис. 9.29), доступні з категорії **Helpers** панелі **Create**.

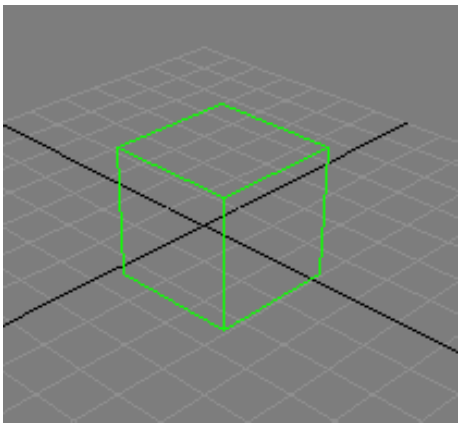


Рисунок 9.29 – Об'єкт типу **Dummy**

Дані об’єкти не відображаються при рендерингу, а у вікнах проєкцій подані у вигляді каркасних кубів. Основне призначення об’єктів **Dummy** – установлення зв’язків між окремими складовими складної ієрархічної структури для організації пов’язаної анімації – тоді при обертанні фіктивного об’єкта будуть обертатися і всі пов’язані з ним інші об’єкти.

Іншим досить поширеним напрямком застосування **Dummy**-об’єктів є анімація націлених камер і джерел світла. В такому випадку об’єкт-пустушка і камера (або джерело світла) зв’язуються в ієрархічну структуру, де камера з мішенню є нащадком фіктивного об’єкта.

Після цього досить буде призначити траєкторію руху об’єкта-пустушки з його автоматичним обертанням відповідно до напряму вектора руху (прапорець **Follow** у згортку **PathParameters** панелі **Motion**), і це забезпечить автоматичне слідування камери за фіктивним об’єктом.

Причому напрямок мішені камери завжди буде відповідати напрямку вектора руху, а сама камера буде слідувати за об’єктом на вказаній відстані, автоматично налаштовуючи позиціонування.

9.6. Матеріали в 3DS MAX

Як відомо, будь-які об’єкти, які нас оточують в реальному житті, мають свій характерний малюнок, за яким ми можемо безпомилково їх ідентифікувати. Створені тривимірні об’єкти спочатку виглядають досить просто і можуть відрізнятися тільки кольором. Щоб наділити об’єкти фізичними властивостями, наприклад прозорістю, шорсткістю, здатністю заломлювати або відображати світло – необхідно для кожного об’єкта сцени встановити характеристики матеріалу.

Крім вбудованого візуалізатора **Scanline** можна встановлювати сторонні рендер-машини, що володіють розширеними функціями обробки світло/тіней і матеріалів. Одним з таких візуалізаторов є **V-Ray**.

V-Ray – потужний інструмент візуалізації, що підтримує **Depth of Field (Глибина різкості)**, **Motion Blur (Ефект «розмиття» в русі)**, **Displacement (Карта зміщення зі збільшенням деталізації тривимірних об’єктів)**. Крім цього, **V-Ray** має власні джерела освітлення, систему сонце – небо для реалістичного освітлення природним світлом, і фізичну камеру з параметрами, аналогічними реальним фото- і відеокамерам.

Система **V-Ray Proxy** дозволяє виробляти прорахунок надзвичайно великих масивів однотипних об’єктів, які перебувають сумарно на десятках

мільярдів полігонів. Убудовані шейдери надають користувачеві широкі можливості для імітації практично будь-яких матеріалів.

V-Ray SDK дозволяє як програмувати власні шейдери, так і адаптувати систему під вирішення специфічних завдань. Можливість прораховувати окремі елементи зображення у вигляді каналів, таких як **Глибина, Дифузний колір, Альфа, Відображення, Заломлення, Тіні** і інших, надає велику свободу постобробленню в пакетах компоузінг і монтажу (рис. 9.30).

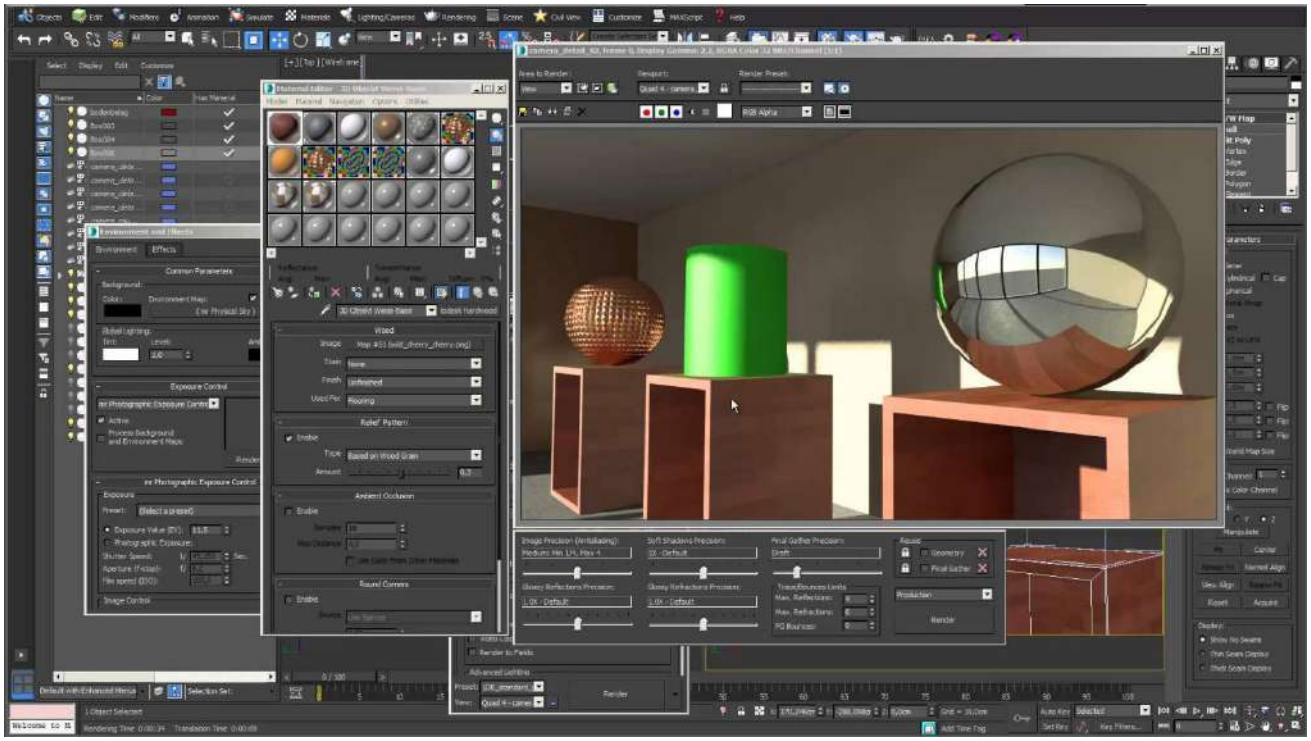


Рисунок 9.30 – Приклад використання різних матеріалів
у сцені 3DS MAX

Будь-який матеріал створюється в **3DS MAX** шляхом завдання трьох основних параметрів, а саме, **Колір (diffuse map)**, наскільки він прозорий **(refraction)**, наскільки від відображає предмети **(Reflection)**, матовий або глянсовий, якатекстура рельєфу. Підключається безпосередньо через карти. Налаштування викликаються через клавішу **M** – матеріал едітор.

9.6.1. Тайлінг (Tiling)

Тайлінг (Tiling) – багаторазове заміщення поверхні текстурою **bitmap** (рис. 9.31).

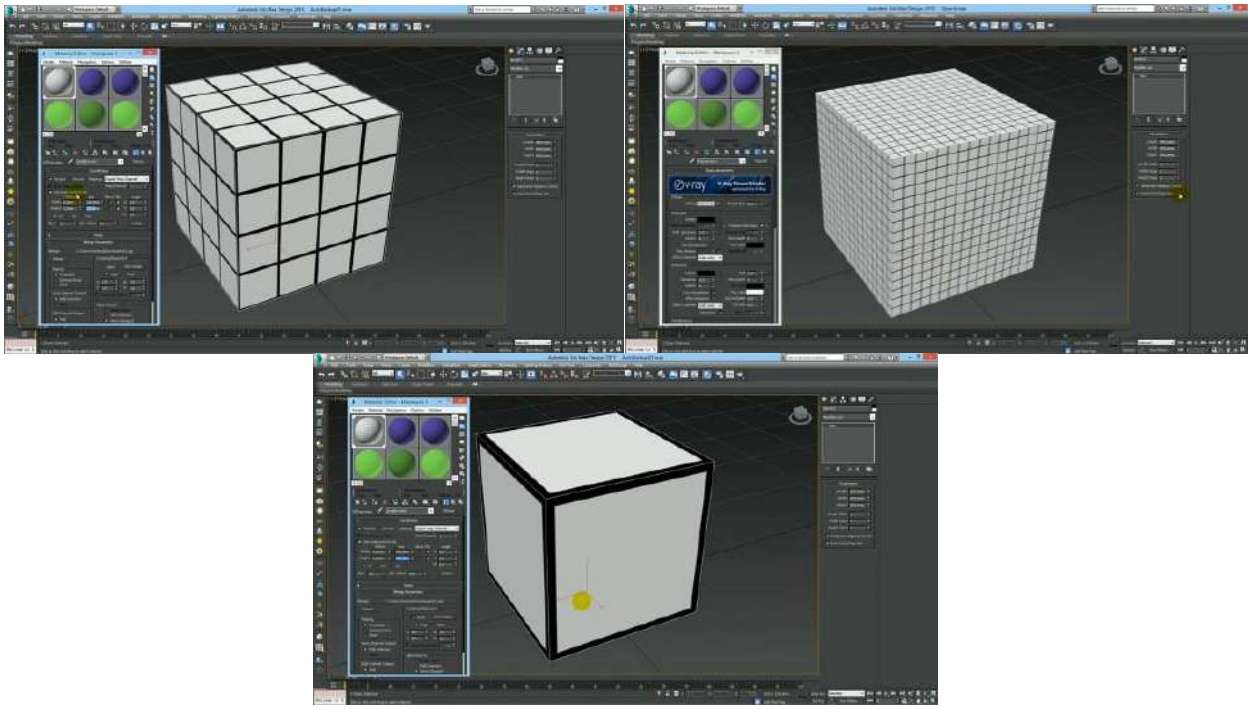


Рисунок 9.31 – Приклад заміщення **Тайлінг**

При виборі кольору ми можемо або вручну задати необхідний відтінок з палітри кольорів, або вибрати заготовки з вбудованих карт матеріалів. Там же, у вкладці дифузних квітів задається параметр шорсткості поверхні **(Roughness)**, задається або значенням від нуля до одиниці, або підключенням відповідної карти.

Refraction – ступінь прозорості об’єкта, як і в налаштуванні відображень, 100 %-й чорний об’єкт є повністю непрозорим, а 100 %-й білий колір вказує на повну прозорість об’єкта.

Наступний параметр – це **Reflection** – ступінь відображення задається або відтінком від білого до чорного, де 100 %-й білий колір відповідає дзеркальній поверхні, а 100 %-й чорний – повністю не відбиваючим поверхням. Також, як і з дифузним кольором, можна задати і колірний відтінок відображення і такожкарту відображення матеріалу.

Карта відображень (reflection maping) – карта відображень поверхні. Задається чорно-білим градієнтом від білого до чорного, де чорний колір вказує відсутність розуміння в конкретній області поверхні і навпаки (рис. 9.32).

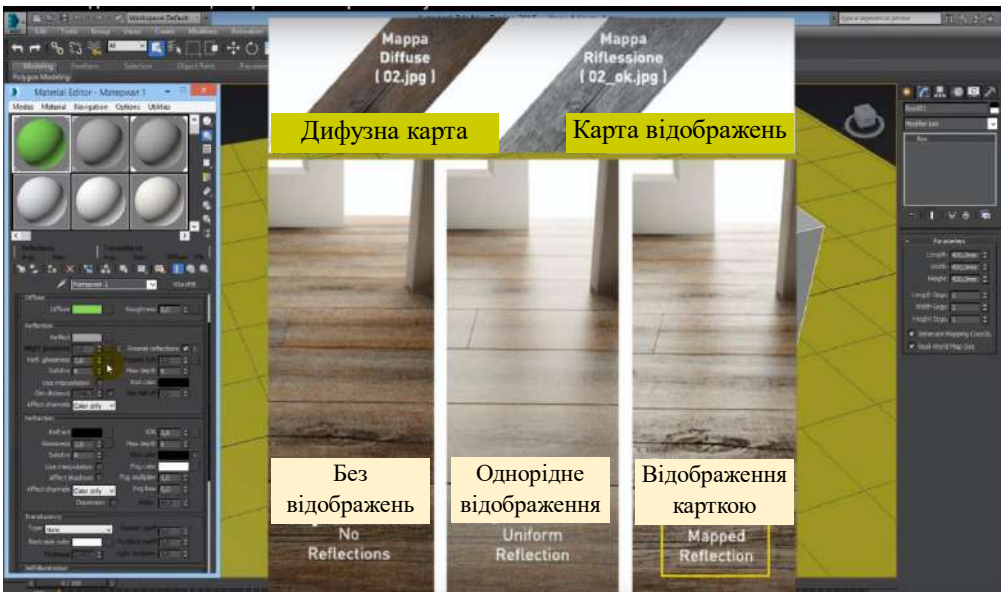


Рисунок 9.32 – Параметри відображень при використанні процедурних карт

Далі в параметрах **Highlight Glossiness** проводять налаштування блиску відображення, тобто відображення можуть бути як з різкими краями відблиску, так і зі змазаними краями відображення.

Reflection Glossiness – розмиття відображення, тобто від 0 до 1 – від розмитої форми відображення, до повністю чіткої межі відображень (рис. 9.33 – 9.34).

Fresnel Reflection – відображення Френеля, сила відображення залежно від кута зору на об’єкт.

Use Interpolation – використання інтерполювання світла. Рекомендується включити цю функцію, так V-Ray зможе швидше прораховувати рендер.

Glossiness – параметр розмитості прозорості, тобто відповідає за глянцевість або матовість поверхні. 1,0 означає, що об’єкт максимально глянсовий, 0 – максимально матовий.

Subdivs – кількість етапів прорахунку – відповідає за якість промальовування відображень.

Affect Shadows – прорахунок тіні, згідно з прозорістю об’єкта.

Fog Color – загасання світла при проходженні через об’єкт. Також вказується відтінок світла, якщо об’єкт підфарбовує його.

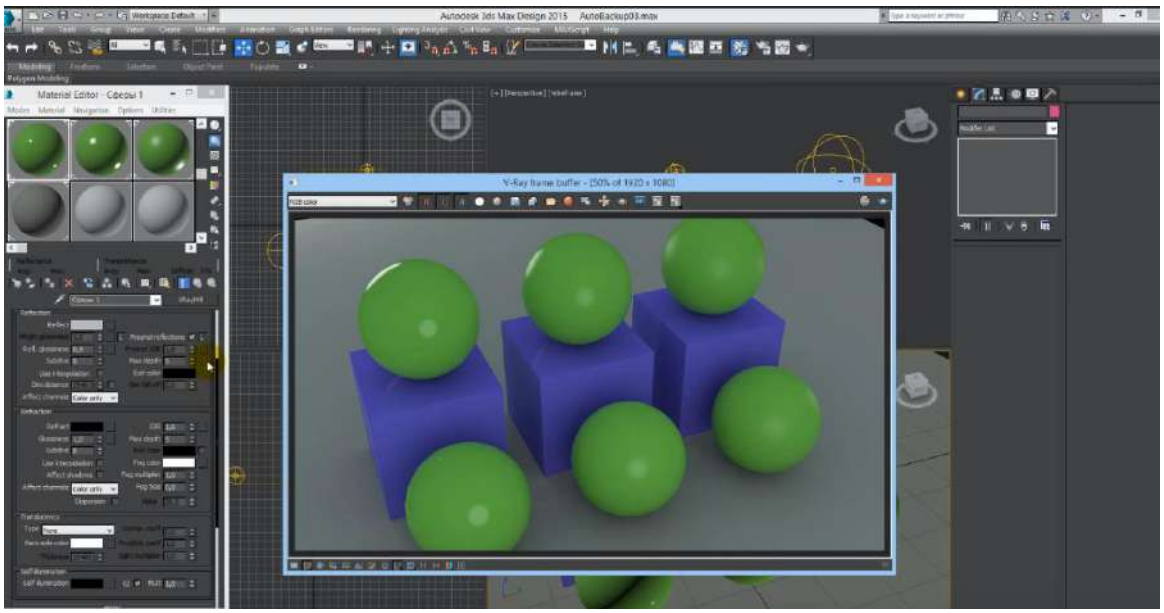


Рисунок 9.33 – Параметри згладжування відблисків поверхонь

Fog Bias – ступінь прозорості тонких деталей об’єкта.

Translucency – підповерхневе розсіювання. **Hard** – для твердих матеріалів. **Soft** – для води. **None** – не застосовується. **Hybrid** – найреальніший алгоритм, використовується для шкіри, молока. **Backside color** – двосторонній матеріал, що копіює **fog color**.

Thikness – параметр товщини стінок об’єкта. Якщо об’єкт порожнистий в примітивах, то цим параметром його можна заповнити і прибрати прорахунок внутрішніх перевідбиттів.

Scatter coef – коефіцієнт розсіювання світла всередині об’єкта. Тобто наскільки сильно світло буде перевідбиватися всередині матеріалу. Нульове значення вказує те, що всі промені будуть розсіюватися у всіх напрямках. Одиниця вказує на те, що промені розсіюються тільки під кутом падіння (входу) в поверхню об’єкта.

Fwr / bck coef – коефіцієнт відповідає за кут напрямку променів розсіювання, 0 вказує на 100 % проходження променів світла всередину об’єкта, 1 – вказує на те, що промені відбиваються назовні об’єкта під кутом падіння.

Light multiplier – підсилювач підпросторових розсіювань. Застосовується, якщо значення попередніх пунктів з заломлення від 0 до одиниці недостатні.

Також матеріали можуть самі випромінювати свічення та заломлювати промені світла, які проходять крізь них.

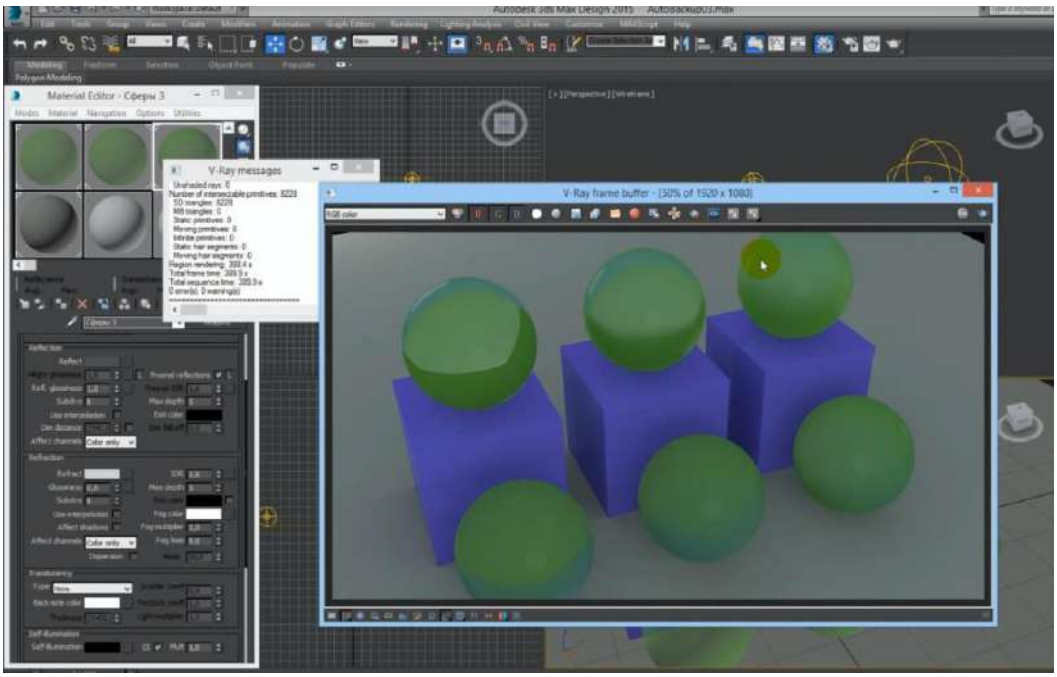


Рисунок 9.34 – Підсумковий рендер включених відображень і заломлень

Self Illumination – власне світіння матеріалу. Вказується крім сили світла його колір.

Каустика – це відблиски світла, які видно на об’єктах сцени після того, як світло було відображене або заломлене через який-небудь скляний / дзеркальний об’єкт. Каустика буває двох видів: рефлективна (базована на світлі, відбитому від об’єктів сцени) і рефрактивна (базована на світлі, що пройшло через заломлення середи).

Рельєфне текстурування **(Bump maping)** (рис. 9.35) – метод текстурування в комп’ютерній графіці для додання більш реалістичного вигляду поверхні об’єктів. Ви можете накладати текстуру, взяту з дифузного bitmap зображення, або спеціально намальованої карти текмтури, або скористатися набором текстур, градієнтів і шумів 3DS MAX. Отриманий рельєф не є рельєфом і поверхня залишається плоскою, але з промальованим світлотіньовим малюнком. Зміна параметрів змішування збільшує і зменшує контраст тіней на поверхні.

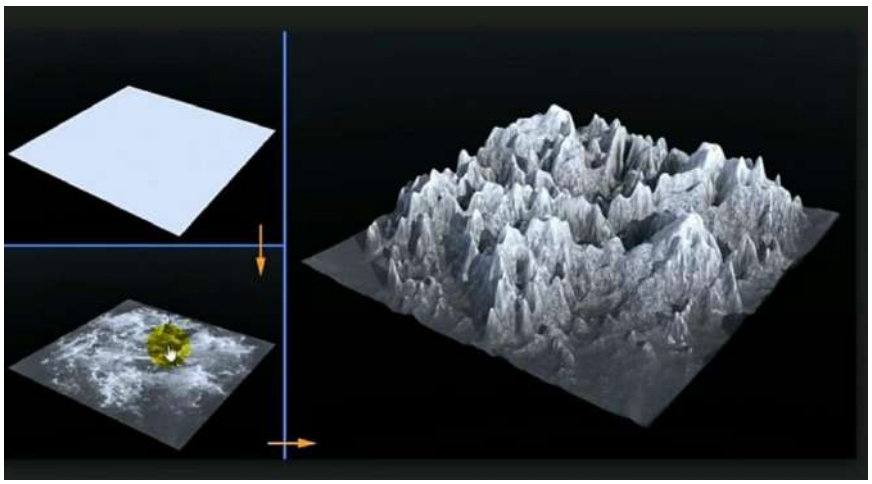


Рисунок 9.35 – Карта рельєфу **(Bump Map)**

Процедурна карта, або ж карта зміщення (**Displacement maping**) – зміна геометричних позицій точок поверхні за рахунок відхилення кожного пікселя згідно з величинами яскравості, заданим у файлі процедурної карти (рис. 9.36 – 9.37). Величина відхилень задається у процедурній карті, яка завантажується аналогічно бампкартам. Градієнтний малюнок поверхні нагадує карту рельєфу і відповідає за ступінь відхилення пікселів від нормалі поверхні.

Рисунок 9.36 – Використання **Displacement maping** для додання вигляду шорсткості поверхні

Рисунок 9.37 – Використання **Displacement maping** для додання тривимірних відбитків

9.7. Контрольні запитання та завдання

1. Що таке полігональне моделювання?
2. Наведіть приклади використання лофтінга при створенні 3D моделі
3. Яким чином проводиться створення реалістичної поверхні та реалістичних властивостей матеріалів?
4. Назвіть всі методики створення нових об’єктів у 3DS MAX.
5. Яку ієрархію об’єктів використовує 3DS MAX?
6. Перелічте всі булеві операції, що існують у 3DS MAX. Наведіть приклади застосування.

9.8. Завдання для самостійного розв’язання

Створити два примітивних об’єкти, за допомогою сплайну та за допомогою існуючих примитивів також створити третій об’єкт, шляхом об’єднання їх операцією Intersection.

10. ОСВІТЛЕННЯ, ТИПИ ОСВІТЛЕННЯ

10.1. Схеми імітації студійного освітлення

У будь-якому редакторі тривимірної графіки реалістичність візуалізованого зображення залежить від трьох основних чинників: якості створеної тривимірної моделі, вдало виконаних текстур і освітлення сцени.

Правильно підібране освітлення є найбільш істотним фактором забезпечення реалізму сцени і об'єктів. Воно створює контрасти між об'єктами, робить використані матеріали більш яскравими і виразними і дозволяє налаштовувати тіні об'єктів. Крім того, освітлення визначає загальний настрій сцени – наприклад, розсіяне світло створює ефект умиротворення, тьмяне освітлення може викликати страх, мерехтливе світло – відчуття тривоги і т.п.

Середа 3DS MAX пропонує нам кілька джерел світла (ДС), коректно працюючих зі стандартним **сканлайн-візуалізатором (Scanline Default Renderer)**. Всі вони розрізняються способом випромінювання світла і, що вже вдругорядне, формою відкидаємої тіні. За допомогою них можна імітувати практично будь-яку схему освітлення, доступну в реальному світі. Всі стандартні ДС доступні в 3DS MAX, повторюють властивості джерел, зустрічающіхся в нашому житті. На даний момент в 3DS MAX доступні: **Target Spot (Націлений прожектор), Free Spot (Вільний прожектор), Target Direct (Націлений прямий ДС), Free Direct (Вільний прямий ДС), Omni (Всенаправлений), SkyLight (Світло небесного купола)**.

10.1.1. Omni (Точкове джерело розсіяного світла)

Omni (рис. 10.1) випромінює світло з точки у всіх можливих напрямках, від єдиного точкового джерела подібно лампочці без абажура. Тіні від предметів, підданих випромінюванню **Omni,** нагадують за формою тіні **Spot**. Це проєкції що розширюються при віддаленні від об'єкта. За допомогою **Omni** можна імітувати світло від **свічки, різних ламп, кульової блискавки** і т.п.

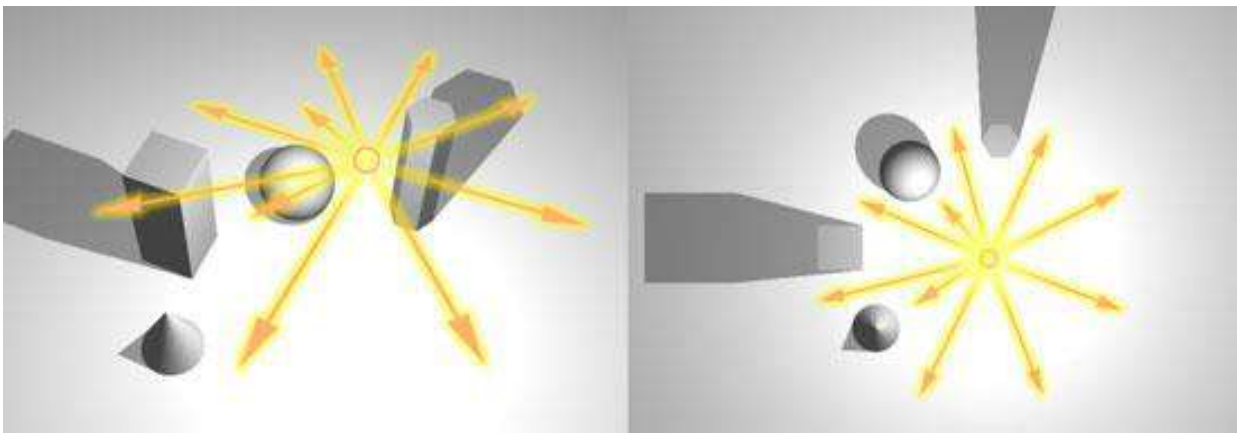


Рисунок 10.1 – Поширення світла і форма тіні у джерела типу **Omni**

10.1.2 Target Spot і Free Spot (Націлений прожектор і вільний прожектор)

Target Spot і **Free Spot** – поширюють промені з точки в певному напрямку конічним потоком і висвітлюють область всередині конуса. Різниця цих двох джерел (рис. 10.2) полягає в тому, що напрям світлових променів в першому з них строго визначено точкою цілі (**Target**), а друге джерело такої точки цілі не має і тому напрямок світлових променів у ньому може змінюватися при обертанні джерела.

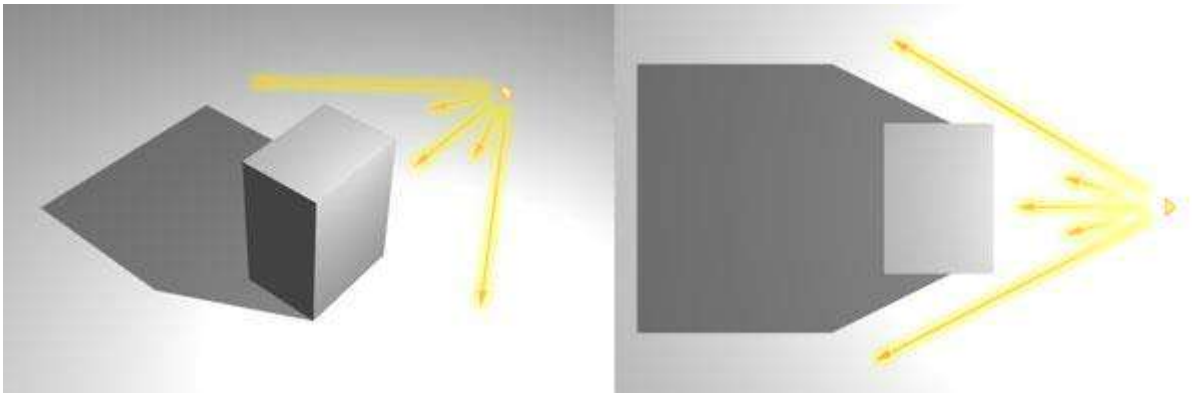


Рисунок 10.2 – Поширення світла і форма тіні у джерела типу **Spot**

Оскільки вихідні промені світла розходяться з точки випромінювання під кутом, то і тінь, яка відкидається цим цим джерелом, нарощує площу в міру віддалення від предмета. Логічно припустити, що для імітації сонячного світла таке ДС не застосовують, але воно добре підійде для штучних ДС. У рідкісних

випадках можна частково імітувати природні ефекти – наприклад, проходження сонячних променів через діри в хмарах або через листя в лісі.

10.1.3. Target Direct і Free Direct (Націлене і вільне джерело, в яких світло поширюється паралельно, за формою паралелепіпеда)

Джерело світла типу **Direct** (рис. 10.3) імітує поширення паралельних променів. Поширюють промені з площини паралельним потоком у певному напрямку і висвітлюють область всередині прямого або похилого циліндрів. Дані джерела розрізняються між собою тим, що напрямок світлових променів у першому з них має прив'язку-ціль, а другий спрямований вільно (напрямок відбитих ним світлових променів змінюється при обертанні джерела).

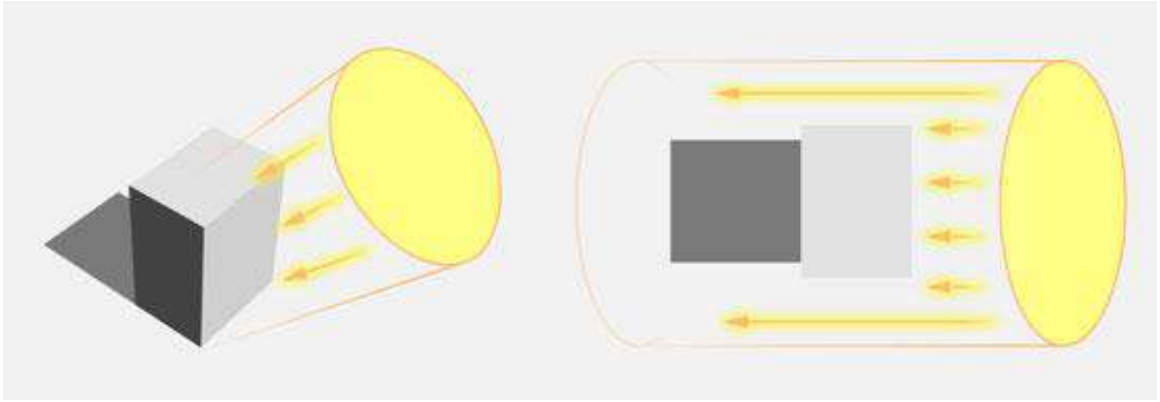


Рисунок 10.3 – Поширення світла і форма тіні у джерела типу **Direct**

Слово **target** означає, що напрямком світла можна керувати спеціальним маркером у вигляді невеликого кубика, джерела **Free Spot** і **Free Direct** його не мають.

10.1.4. Skylight (Джерело світла в якому його уявні промені не виходять з якоїсь однієї точки)

Уявіть собі гігантський купол (рис. 10.4) над вашою сценою в 3DS MAX, з поверхні якого випромінюється світло. Промені багаторазово перевідбиваються від об'єктів і нарешті згасають, втрачаючи енергію. Це всебічне освітлення з багаторазовими перевідбиваннями.

У підсумку ми отримуємо рівну картину освітлення і м'які тіні (рис. 10.5). Саме так діє Skylight. Таке освітлення часто є самодостатнім і не потребує додаткових ІС. Такий метод освітлення виключає чорні провали в тінях, а отже, втрату деталей.

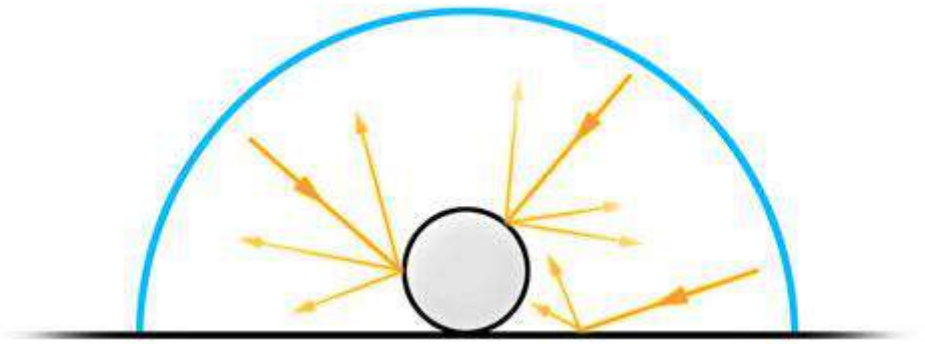


Рисунок 10.4 – Поширення світла і форма тіні у джерела типу **Skylight**

Рисунок 10.5 – М’які тіні при освітленні **Skylight**

За створення джерел світла відповідає категорія **Lights (Джерела світла)** панелі **Create (Створити)**, при виборі якої стають доступними всі описані вище типи джерел (рис. 10.1–10.5). Технологія їх створення нагадує створення об’єктів геометрії. Будь-яке джерело світла можна переміщати, повертати і масштабувати на видових екранах так само, як і всі інші стандартні об’єкти.

Параметр **Multiplier** у вкладці **Intenity / Color / Attenuation** впливає на силу джерела світла.

Параметр **Dens у вкладці Shadow Parameters** впливає на **густину тіні.**

Для того щоб **межа світла –– тіні** була **м’якою**, у вкладці **Spotlight Parameters** поставте значення **Hotspot/Beam** в кілька разів менше значення **Falloff/Field,** зміни ви будете спостерігати у вікні проєкції.

Так само **на тінь впливає відстань до джерела світла**. Чим воно **далі, тим більш розмитою** буде тінь, **чим ближче – тим тінь буде чіткіше.**

Для створення кінематографічної картинки, яка буде ефектно виглядати, необхідно використовувати кілька джерел світла у сцені, а також враховувати розсіювання, відбиття і поглинання світла навколишніми поверхнями.

10.2. Схеми імітації студійного освітлення

Існує безліч схем освітлення але всі вони є одним з варіантів схеми поданої на рис. 10.6.

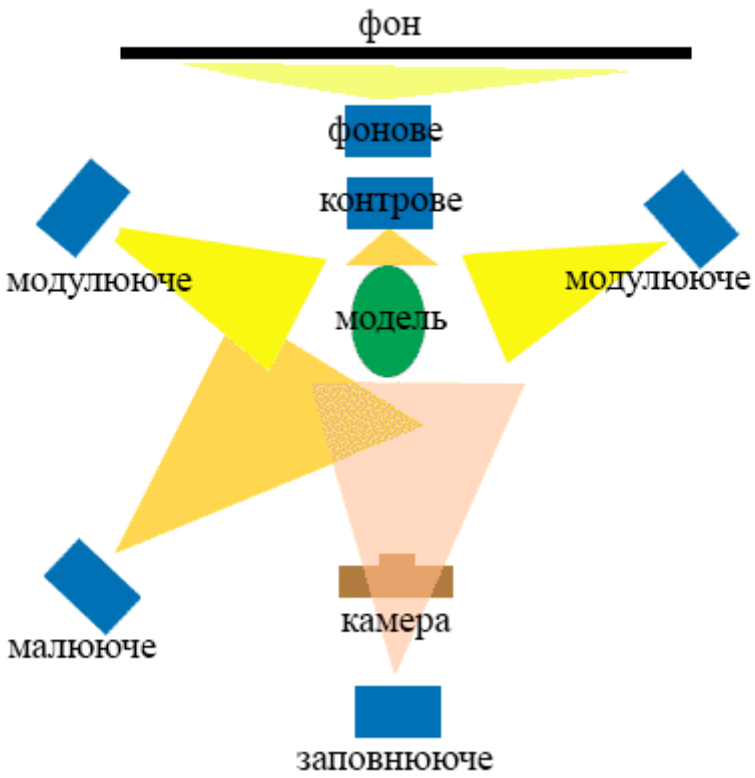


Рисунок 10.6 – Поширені схеми розстановки джерел світла

Малююче світло – основне джерело світла, що створює світлотіньовий малюнок. Всі інші джерела відносно до нього другорядні і служать для підкреслення окремих деталей і висвітлення тіней.

Заповняюче світло – джерело, світло від якого йде від місця, де встановлена камера. Служить для висвітлення тіней від джерела Малюючого світла. Можливий варіант, коли Малююче та Заповнююче джерела суміщені, тоді виходить безтіньовий малюнок.

Фонове – служить для освітлення фону. Оскільки часто джерело світла неможливо помістити за об’єктом, то він зсувається убік, або ставляться два джерела з боків.

Контрове – служить для освітлення об’єкта зйомки ззаду.

Моделююче – служить для виділення світлом бічних поверхонь об’єкта зйомки.

Всі разом ці джерела світла застосовуються досить рідко. Для більшості

видів зйомки досить трьох (малююче, заповнююче і фонове) або двох (малююче і фонове) джерел. Завжди слід мати на увазі, що ряд джерел освітлення може бути замінений звичайними відбивачами.

Наприклад, джерело заповнюючого світла може бути замінено білим відбивачем, який стоїть поруч з об'єктом зйомки. Також слід розуміти, що світлотіньовий малюнок – це просто співвідношення яскравості ділянок, і якщо ми хочемо, щоб щось виглядало більш світлим, то можна не висвітлювати додатково ці місця, а затінювати іншу частину об'єкта, застосовуючи, наприклад, чорний прапор.

Флаг – це поверхня чорного кольору, поглинаюча світловий потік, розташовується таким чином, щоб затемнити частину світла, яка надходить від освітлювачів.

Для того щоб світлотіньові переходи були більш різкими або більш м'якими, застосовуються різні насадки на джерела світла. Існує загальне правило – чим більше поверхня випромінювання, тим м'якше тіні. Для отримання м'яких тіней використовуються софт-бокси різної величини й розсіювачі, а для отримання різких тіней, наприклад сотові насадки. Найм'якіші тіні виходять при освітленні світлом з великого вікна в похмуру погоду.

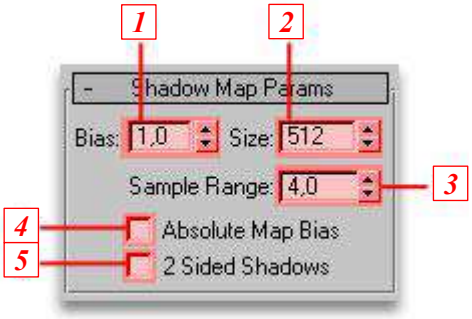
10.3. Налаштування тіней

Серед стандартних джерел світла три джерела (а саме **Spot**, **Direct** і **Omni**) дозволяють вибрати тип прораховуваних тіней. Якщо використовувати штатний стандартний візуалізатор **Default Scanline Renderer (DSR)**, то інтерес для нас будуть скаладати: **Advanced ray-traced shadows**, **Area shadows**, **Ray-traced shadows**, **Shadow maps**.

При виборі типу тіні серед згорків параметрів ДС з'являється згорток параметрів тіні, назва якого починається з найменування типу.

10.3.1. Shadow Map

Найпростіший і невибагливий до розрахункових ресурсів тип тіней (рис. 10.7).



1. Відстань об'єкта від падаючої тіні.

2. Розмір карти, на основі якої розраховується тінь. Чим більше карта, тим якісніше розраховується тінь. Краще використовувати числа порядку 2^n.

3. Розмиття кромки тіні. Збільшення параметра дозволяє позбутися від зубчастого краю кромки при низькій роздільній здатності карти.

4. Параметр, який відповідає за керування значенням ***Bias****. За замовчуванням вимкнено (найкращий результат у більшості випадків). У випадку з анімацією може допомогти включення опції.*

5. Якщо вимкнено, то світло проходить через поверхню, якщо натикається на полігони, відвернені від нього нормалями. Включення опції дозволяє отримати коректні тіні.

Рисунок 10.7 – Налаштування параметрів відображення тіней у 3DS MAX

На рис. 10.8 верхній ряд зображень наочно показує зміну якості тіні при збільшенні параметра **Size**. Навіть значне збільшення розміру карти не вирішує проблеми зубців на краях тіні, хоча безумовно малюнок тіні стає більш пророблений.

У другому ряду у всіх трьох випадках розмір карти залишається однаковим, але змінюється параметр **Sample Range**. Поступово збільшуючи його, ми позбулися зубчастості, розмивши кромку тіні.

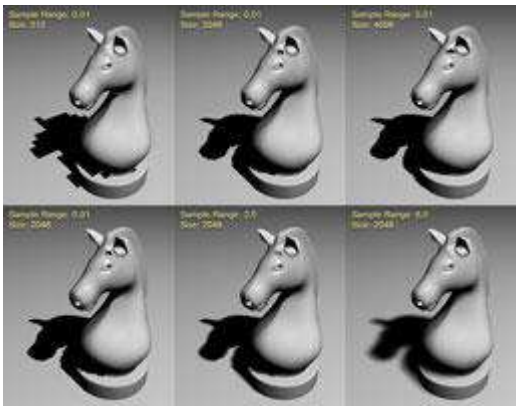
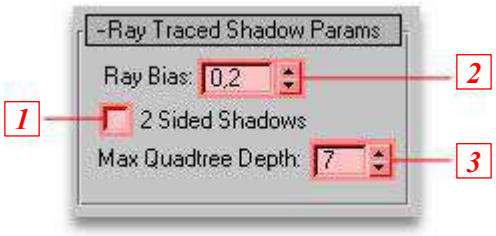


Рисунок 10.8 – Зміна якості тіні типу **Shadow Map** при різних параметрах

10.3.2. Ray-Traced Shadows (Тіні з трасуванням променів)

Тіні цього типу розраховуються на основі алгоритму трасування, мають чіткі краї (рис. 10.9 – 10.10) і практично не піддаються налаштуванню.

Тінь **Ray-Traced Shadow** (рис. 10.7) точніша за **Shadow Map**. До того ж вона здатна враховувати прозорість об'єкта, але тим часом «суха» і чітка, що виглядає не дуже природно в більшості випадків. **Ray-Traced Shadow** більш вимоглива до ресурсів комп'ютера, ніж **Shadow Map**.



1. Відстань об'єкта від падаючої тіні.

2. Якщо вимкнено, то світло проходить через поверхню, якщо натикається на полігони, відвернені від нього нормалями. Включення опції дозволяє отримати коректні тіні.

3. Глибина трасування – параметр, який відповідає за опрацювання тіні. Збільшення цього значення може значно збільшити час візуалізації.

Рисунок 10.9 – Налаштування опрацювання контуру тіней у 3DS MAX

Ray-Traced Shadows з ДС типу **Omni** займуть більше часу на візуалізацію, ніж зв'язка **Ray-Traced Shadows + Spot** (або **Directional**).

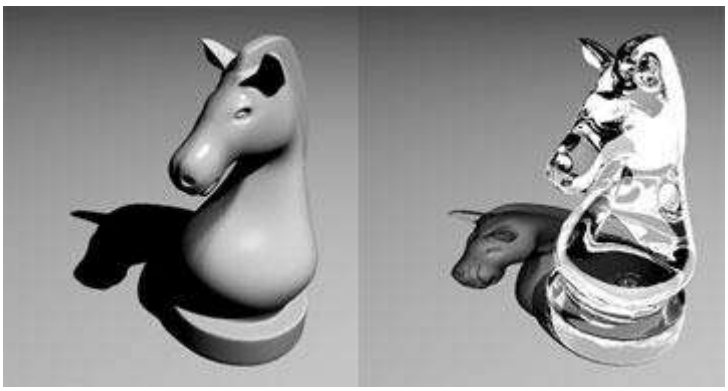


Рисунок 10.10 – Тіні **Ray-Traced Shadows** від непрозорого і прозорого об'єктів

В налаштуваннях тіней також необхідно вказувати тип джерела світла, а саме:

– ***Simple*** – одиночний промінь виходить з ДС. Тінь не підтримує будь-якого згладжування і налаштування якості.

– ***Rectangle Ligh***t – імітується випускання світла з прямокутної області.

– ***Disc Light*** – ДС поводиться так, як нібито воно придбало форму диска.

– ***Box Light*** – імітація кубічного ДС.

– ***Sphere Ligh***t – імітація сферичного ДС.

10.4. Віртуальні камери у 3DS MAX

Віртуальні камери – спеціальний інструмент в 3DS MAX, працюючи з яким, ми вказуємо точку, звідки будемо спостерігати сцену, напрямок, кут огляду, ракурс і т. д.

Ра́курс – це точка зору на об’єкт у просторі (рис. 10.11), а також одержувана в процесі зйомки проєкція (зображення, яке фіксується на полотно, фотоплівку або цифрову матрицю фотоапарата) об’єкта в даній точці.

Фактично наше вікно проєкцій **Перспектива** – це і є камера, яка присутня в сцені.

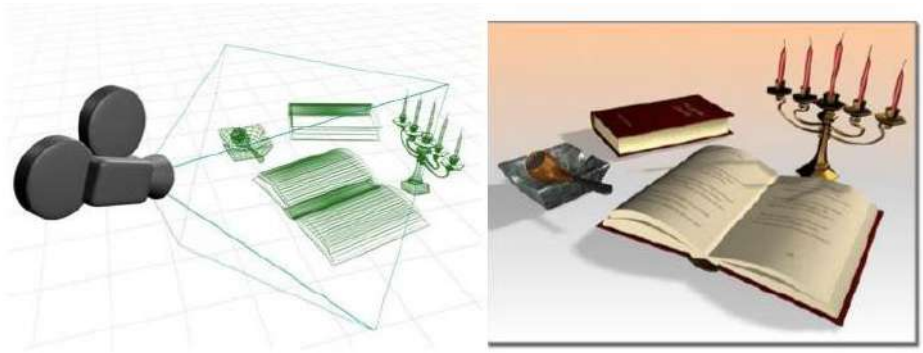


Рисунок 10.11 – Ракурс і проєкція віртуальної камери

Камери в **3DS MAX** є додатковими об’єктами. Тому з ними, так само як і з джерелами світла, можна виконувати всі перетворення, наприклад, переміщення, обертання, масштабування. Аналогічно освітлювачам, з камерами можна працювати в чорновій сцені, але у фінальній сцені їх не буде видно, вони там не промальовані.

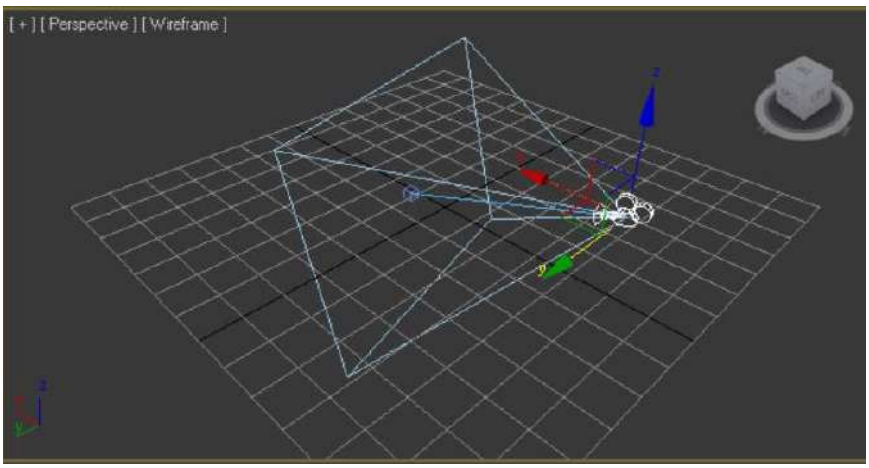


Рисунок 10.12 – Націлена **Target Camera** у вікні проєкцій

Існують камери двох видів: **Target** і **Free**.

Камера Free є вільною (без можливості прив'язки до об'єкта зйомки).

Камера Target (рис. 10.12) є спрямованою камерою і складається з власне камери і мішені. Перше клацання створює власне камеру, далі, при натиснутій лівій клавіші, покажчик переміщається в ту точку простору, в якій необхідно встановити мішень.

Мішень камери **Target** при необхідності можна пов'язувати з об'єктом. Це буває зручним при мультиплікації, наприклад, коли переміщаючись, камера весь час тримає об'єкт у полі зору. Також для спрямованої камери можна вказати фокусну відстань за допомогою параметра **Target Distance (Фокусна відстань)**, що використовується при створенні ефекту глибини різкості.

Спрямовану або вільну камери можна легко анімувати, точно так же, як це робиться з будь-яким об'єктом **3DS MAX**. В результаті ви отримаєте кінематографічну, динамічну зйомку.

Об'єкти типу **Cameras (Камери)** мають ті ж характеристики, що й справжні камери – **Lens (Розмір фокусної відстані)** і **Field of View (Поле зору)**.

10.4.1. Налаштування віртуальних камер

Апертура (aperture) – це отвір в діафрагмі об'єктива, який регулює кількість світла, що надходить в камеру і потрапляє на плівку або цифрову матрицю. Апертура регулюється за допомогою діафрагми. Діафрагма діє подібно зіниці ока, відкриваючи або закриваючи об'єктив камери і тим самим регулюючи кількість світла, що проходить в камеру. Величина отвору характеризується діафрагмовим числом. Велике діафрагмове число означає

невелику апертуру, яка пропускає мало світла в камеру. І навпаки, менше діафрагмове число пропускає менше світла в камеру. У цифровій зйомці зміна діафрагми використовується для ефекту малої глибини різкості (рис. 10.13). За замовчуванням тривимірні картинки мають дуже чіткий вигляд. Щоб виправити цей недолік, потрібно приділити особливу увагу глибині різкості

Рисунок 10.13 – Приклад впливу значення діафрагми на відображення об’єктів в кадрі

DOF (depth of field)

Велика глибина різкості робить все зображення різким і чітким. А якщо зменшити глибину різкості, то різкою залишиться лише та частина кадру, яка розташована поблизу фокусу, а все інше зображення буде розмито. Цей ефект можна використовувати, коли нам необхідно зосередити увагу глядача на якомусь певному об’єкті сцени.

FOV (field of view)

Фокусна відстань – відстань від головного фокуса лінзи (фокусної точки) до її оптичного центру. Зазвичай фокусна відстань вимірюється в міліметрах (рис. 10.15). Чим коротше фокусна відстань об’єктива, тим більше його поле або кут зору **(angle of view)**.

І навпаки, чим довше фокусна відстань об’єктива, тим менше його кут зору. Тому прийнято називати об’єктиви ширококутними і вузькокутовими (довгофокусними). В 3DS MAX в налаштуваннях камер передбачені спеціальні кнопки зі стандартними фокусними відстанями (рис. 10.14).

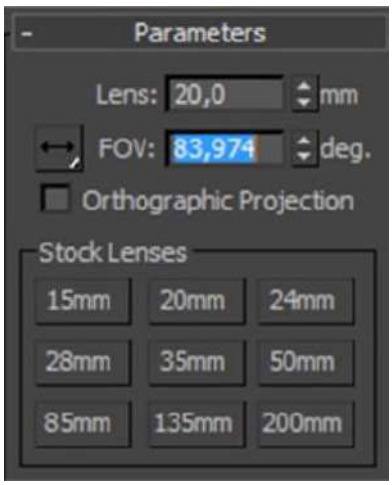


Рисунок 10.14 – Параметри налаштувань фокусної відстані / кута огляду віртуальної камери

Фокусні відстані поширених об'єктивів

Риб'яче око	7.5 мм
Надширококутний	18 мм
Ширококутний	28 мм
Середнєширокий	35 мм
Стандартний	50-55 мм
Середнєфокусний	80 мм
Довгофокусний (телеоб'єктив)	135-260 мм
Наддовгофокусний (супертелеоб'єктив)	500 мм і вище

Рисунок 10.15 – Класифікація об’єктивів за фокусною відстанню

Ці дві характеристики пов’язані між собою, тому при зміні одного параметра другий змінюється автоматично, при цьому, чим більше фокусна відстань камери, тим менше поле зору і навпаки.

Застосування над ширококутних об’єктивів настільки сильно підкреслює різницю масштабів окремих об’єктів або їх частин в кадрі, що це часто стає «неприродним». Людське око здатне охопити простір з великим кутом зору. Охоплюючи дуже великий простір, воно має в той же час незначне поле різкого зображення і, щоб розглянути протяжний об’єкт повністю, змушене «ковзати» уздовж цього об’єкта, розглядаючи його по частинах, і складати уявлення про об’єкт зйомки як суму одиничних вражень. Повноцінна «картинка» складається як сукупність окремих відчуттів тільки в нашій свідомості.

10.5. Перспектива і ракурс зображення

Перспекти́ва (фр. *perspective* від лат. *perspicere – дивитися крізь*) – техніка зображення просторових об'єктів на будь-якій поверхні згідно з тими удаваними скороченнями їх розмірів, змінами обрисів форми і світлотіньових відносин, які спостерігаються в натурі. Іншими словами, це образотворче спотворення пропорцій і форми реальних тіл при їх візуальному сприйнятті. Наприклад, дві паралельних рейки здаються збігаючимися в одну точку на горизонті. Це є спосіб зображення об'ємних тіл, передаючий їх власну просторову структуру і розташування в просторі.

Ми з вами прекрасно знаємо, що реальний світ тривимірний і всі предмети і об'єкти зйомки мають три виміри. Вони об'ємні і також знаходяться в тривимірному просторі. Фотографія передає лише два виміри – висоту і ширину. Третій вимір – глибина простору – в фотографії прямо не передається. Справа в тому, що ми, володіючи бінокулярним зором, отримуємо інформацію про різну віддаленість предметів і об'єктів за допомогою двох очей. Два зображення, що відрізняються одне від одного, в силу того, що вони знаходяться на деякій відстані одне від одного (так званий стереобазою), і одержувані індивідуально кожним оком, тільки у свідомості об'єднуються в цілісну просторову картину (рис. 10.16).

А тепер спробуйте одне око закрити – як сильно зміниться наше відчуття про глибину і протяжність простору... Різна віддаленість предметів, тільки що зараз була більш ніж очевидна, і раптом стає сумнівною. Об'єкти, що мають однаковий розмір, здаються нам тим менше, чим далі від нас вони знаходяться. Паралельні лінії, спрямовані від нас в глибину простору, прагнуть зійтися в одній точці, що знаходиться в нескінченності.

Рисунок 10.16 – Приклад паралельних ліній в горизонті, що сходяться

10.5.1. Ракурс

Ра́курс – це точка зору на об’єкт в просторі, а також, проєкція об’єкта в даній точці, що отримується в процесі зйомки.

Застосовувати нижній або верхній ракурс потрібно залежно від творчого задуму та ідеї, яку ви бажаєте передати цим знімком.

10.5.2. Нижній ракурс

Наприклад, використовуючи нижній ракурс в репортажі, можна візуально посилити висоту стрибка спортсмена. Застосування нижнього ракурсу дозволяє продемонструвати велич і грандіозність того, що знімається, посилити значимість його розміру і підкреслити виразність (рис. 10.17).

Рисунок 10.17 – Приклад використання нижнього ракурсу

При нижньому ракурсі предмет зйомки наче б завалюється вперед або назад, що надає знімку динамічності. Ефект динамічності і руху можна значно підсилити, використовуючи діагонально побудовану композицію.

10.5.3. Верхній ракурс

Зйомка з великим кутом змінює реальні розміри об’єкта, як ніби їх придавлює до поверхні. За допомогою верхнього ракурсу можна відобразити значний простір і розташування в ньому об’єктів,що знімають, а також підкреслити положення важливого об’єкта в загальному просторі композиції (рис. 10.18).

Рисунок 10.18 – Приклад використання верхнього ракурсу

При зйомці в студії верхній ракурс дозволяє зробити оригінальні кадри, проте в цьому випадку потрібно пам’ятати про можливість спотворення звичних пропорцій.

При використанні центрального ракурсу будується фронтальна композиція (рис. 10.19), в якій збігаються вісь знімка і оптична вісь об’єктива камери. В результаті цього предмет зйомки стає наче б то плоским, знімок позбавляється глибини.

Рисунок 10.19 – Приклад центрального ракурсу

На реальному відеоматеріалі часто можна помітити деякі особливості, обумовлені конструкцією камери. На об’єктах реальної камери іноді присутні відблиски, певної форми спотворення, обумовлені геометрією лінз в об’єктиві.

Щоб візуалізоване зображення якомога більше нагадувало справжнє, можна використовувати ефект, що імітує подібні артефакти. У середовищі 3**DS MAX** можна створити такі ефекти камер:

- **Lens Effects** (Відблиски-засвітки);
- **Color Balance** (Баланс білого);
- **Depth of Field** (Глибина різкості, f-stop);
- **Film Grain** (Зернистість);
- **Motion Blur** (Розмиття руху).

Візуалізатор **Mental ray 3.3** має мінімальну кількість параметрів для керування ефектом глибини різкості. Якщо в сцені використовується камера, налаштування ефекту здійснюється за допомогою параметра **f-Stop (Величина апертури)** в згортку **Depth of Field Parameters (Параметри ефекту глибини різкості)**. Параметр **f-Stop (Величина апертури)** визначає кількість світла, що потрапляє в камеру. Тому якщо значення числа діафрагми невелике, виходить невелика глибина різкості, при якій буде чітко видно лише деякі об’єкти.

10.6. Контрольні питання та завдання

1. Назвіть основні типи джерел світла, що використовуються в 3DS MAX.
2. Визначте особливості налаштування окремих типів джерел світла.
3. Опишіть вплив фокусної відстані об’єктива на відображення пропорцій об’єкта в сцені.
4. Наведіть основні схеми моделювання освітлення в сцені та особливості використання.
5. Назвіть основні типи віртуальних камер і об’єктивів, що використовуються у 3DS MAX.
6. Наведіть особливості налаштування відображення тіней і заломлень поверхонь в 3DS MAX.

10.7. Завдання для самостійного розв’язання

Побудувати схему освітлення з 4 різніх джерел освітлення та виконати рендер сцени.

11. АНІМАЦІЯ ПЕРСОНАЖІВ

Визначена анімація – створення ілюзії руху об’єктів за допомогою зміни його форми за заданими наперед значеннями.

Процедурна анімація (англ. *procedural animation*) – вид комп’ютерної анімації, який автоматично генерує анімацію в режимі реального часу відповідно до встановлених правил, законів і обмежень. На відміну від визначеної анімації, коли аніматор вручну визначає кожен кадр і всі параметри створюваної анімації, при процедурній анімації результат може бути в деякій мірі непередбачуваний і при кожному запуску може генерувати різноманітну анімацію.

11.1. Створення скелету персонажа

Для створення анімації в 3DS MAX вже є вбудований елемент **CAT objects** у вкладці **Helpers** (рис. 11.1–11.2). Він дозволяє обрати вже готові скелети різноманітних персонажів, в тому числі і людиноподібних. Також налаштування дозволить модифікувати будь-який з готових скелетів під свої потреби, тобто додати додаткові ноги, руки або голову.

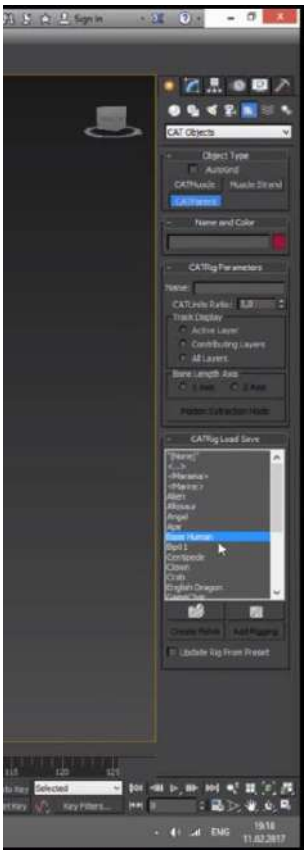


Рисунок 11.1 – Вибір готових віртуальних персонажів у 3DS MAX

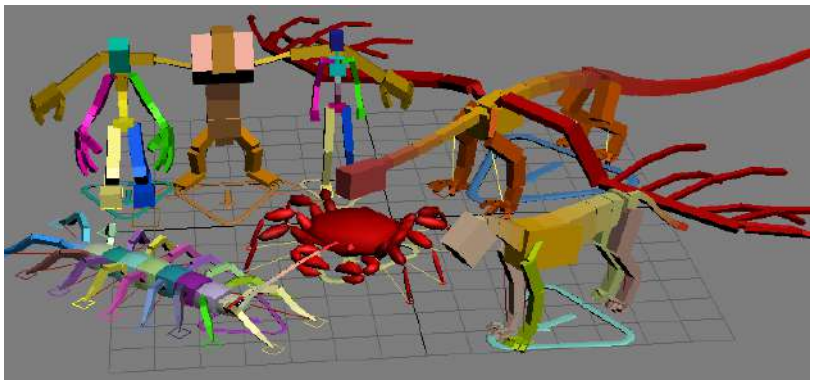


Рисунок 11.2 – Приклад скелетів різноманітних персонажів

Заготівка для людиноподібного персонажа має назву **Base Human** (рис. 11.1). Отриманий скелет повторює основні кістки людини і дозволяє відтворити всі вигини і повороти всіх існуючих суглобів. Кожна кістка **(Bones)** є примітивом і з нею можна виконувати всі дії, пов'язані зі зміною розміру і поворотом, для того, щоб підігнати розмір заготовки до форми анімованого персонажа (рис. 11.3).

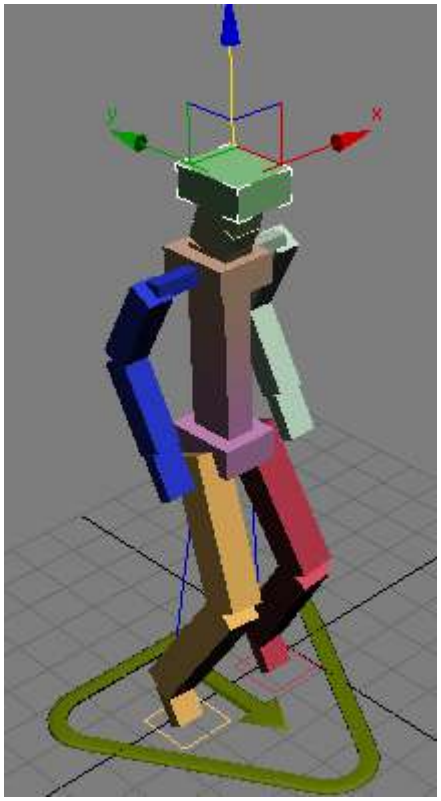


Рисунок 11.3 – Скелет анімованого персонажа

Наступним кроком є підгонка кожної кістки скелета під розміри частин тіла моделі, для чого необхідно, щоб кожна кістка кріпилася в суглобі нашої майбутньої моделі. Таким чином у подальшій анімації ноги і руки персонажа будуть згинатися максимально реалістично. Рекомендується зробити модель напівпрозорою та вимкнути її виділення, щоб оперувати тільки з **CATbones** (рис. 11.4 – 11.5).

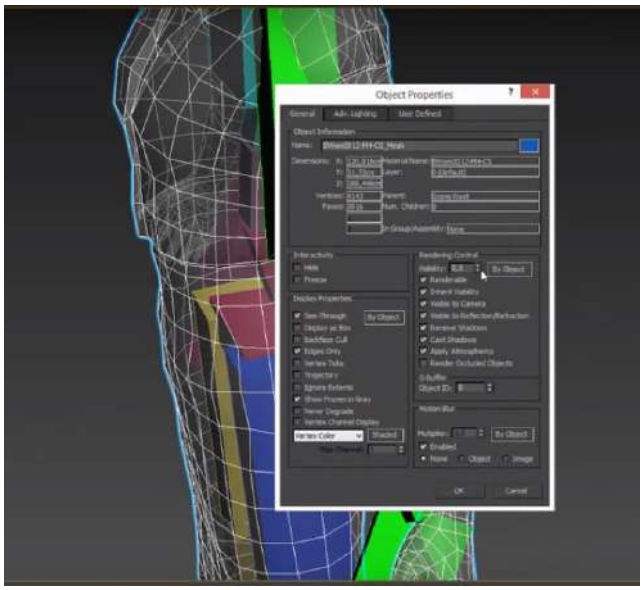


Рисунок 11.4 – Налаштування об’єкта

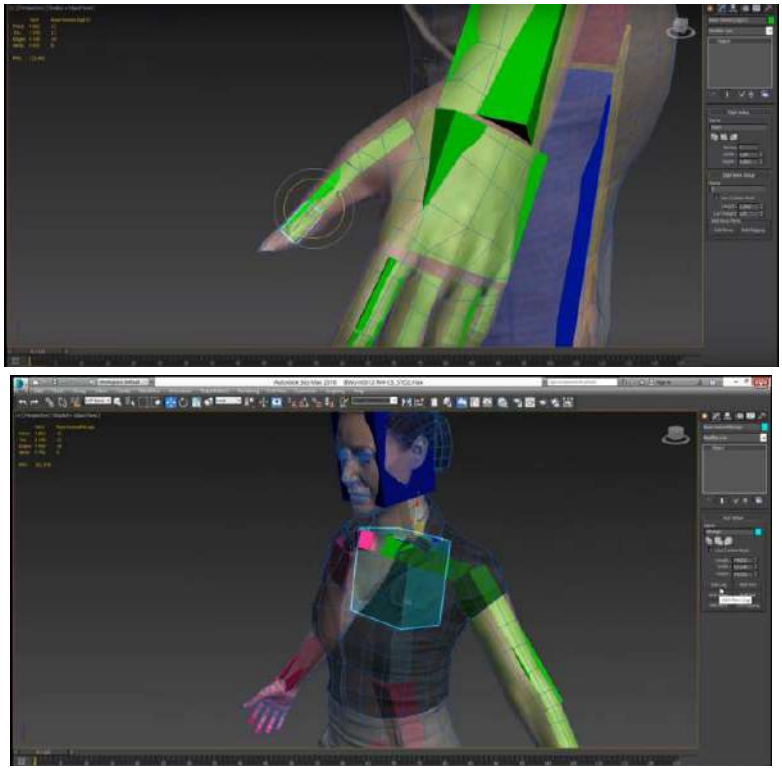


Рисунок 11.5 – Налаштування напівпрозорої оболонки персонажа

Аналогічно кісткам скелета відтворюються **Bones** і для інших рухомих частин людини, навіть якщо в реальному житті там кісток немає, наприклад, для анімації повороту очей (рис. 11.6), відкриття повік і т. д.

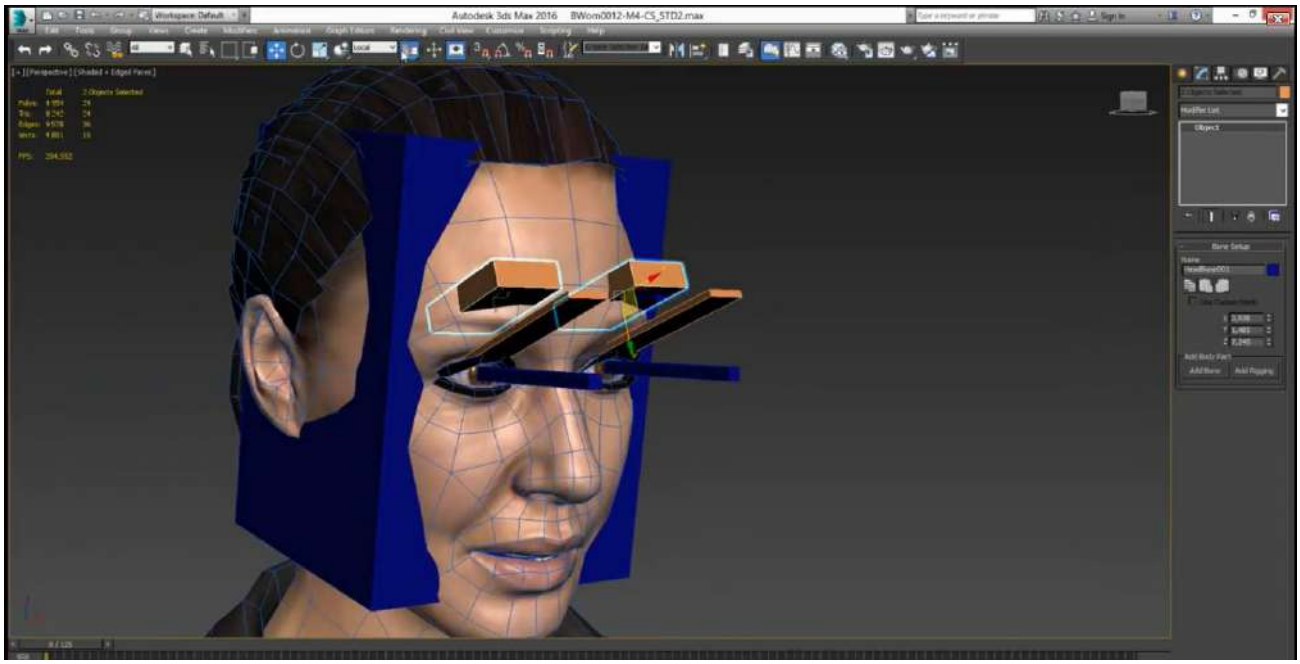


Рисунок 11.6 – Анімація повороту очей

Важливо враховувати, що розмір кісток не такий важливий, як точність кріплення суглобів, тому для анімації напрямку повороту очей, зручніше зробити примітиви такого розміру, щоб вам було зручніше орієнтуватися в їхньому напрямку. Сам скелет не несе будь-якої естетичної складової, а виконує тільки функціонал передачі руху.

11.2. Скінінг

Як тільки розміри кісток скелета підігнані під модель (рис. 11.7), необхідно виконати прив’язку до поверхні тіла моделі. Синхронізація виконується досить просто, за допомогою модифікатора **Skin**, який призначається на об’єкт з сіткою поверхні персонажа. Далі в налаштуваннях модифікатора за допомогою кнопки **Add** (рис. 11.8) завантажуються створені кістки скелета моделі.

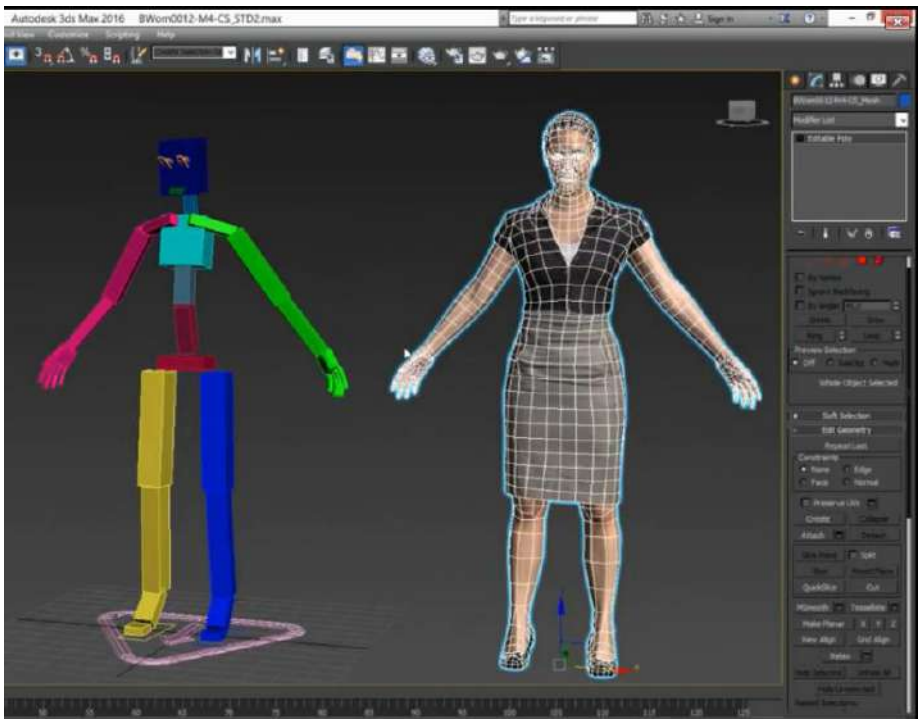


Рисунок 11.7 – Модель людини (справа) і її скелет (зліва)

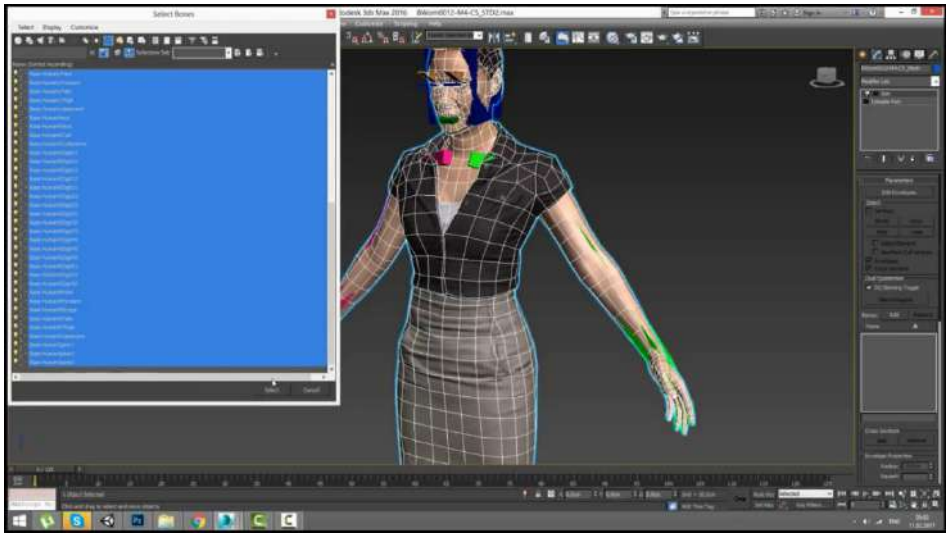


Рисунок 11.8 – Завантаження створених кісток скелета моделі

За замовчуванням не всі кістки можуть вірно закріпитися і адекватно обертати сітку моделі, для перевірки цього необхідно запустити тестову анімацію ходи або бігу персонажа. Таким чином можна контролювати, наскільки вірно згинаються суглоби моделі, чи не переломлюється полігональна сітка поверхні (рис. 11.9).

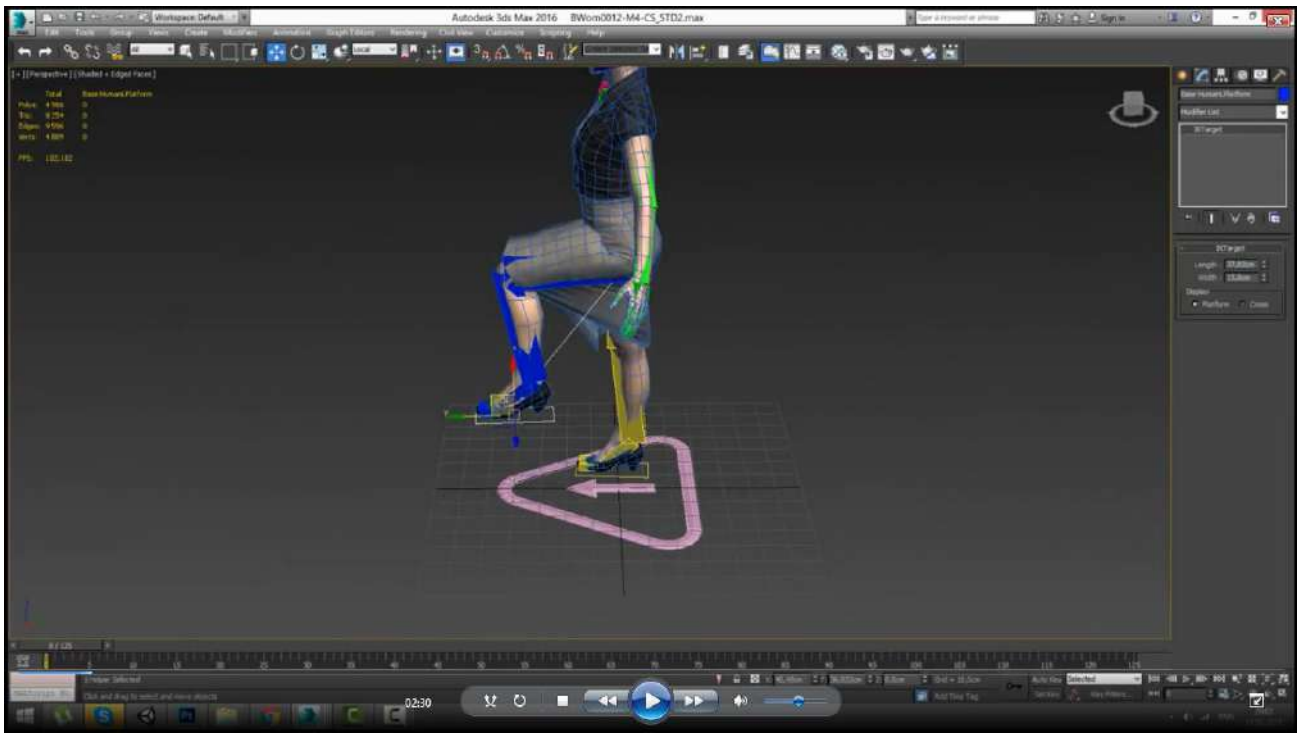


Рисунок 11.9 – Переломлення сітки моделі

У вкладці анімації доступно створення великого числа анімаційних шарів, де в подальшому задається переміщення об’єктів за ключовими кадрами. Це необхідно, тому що кожен персонаж, наприклад для комп’ютерної гри, повинен мати свою анімацію ходи, бігу, стрибка і т. д. Також анімація руху очей, вік, може задаватися на окремому анімаційному шарі.

За замовчуванням до запуску анімації персонаж знаходиться в стані спокою (рис. 11.10). Для того щоб перевірити правдоподібність анімації персонажа і викривлення полігональної сітки, необхідно завантажити встановлену імітацію бігу персонажа (рис. 11.11).

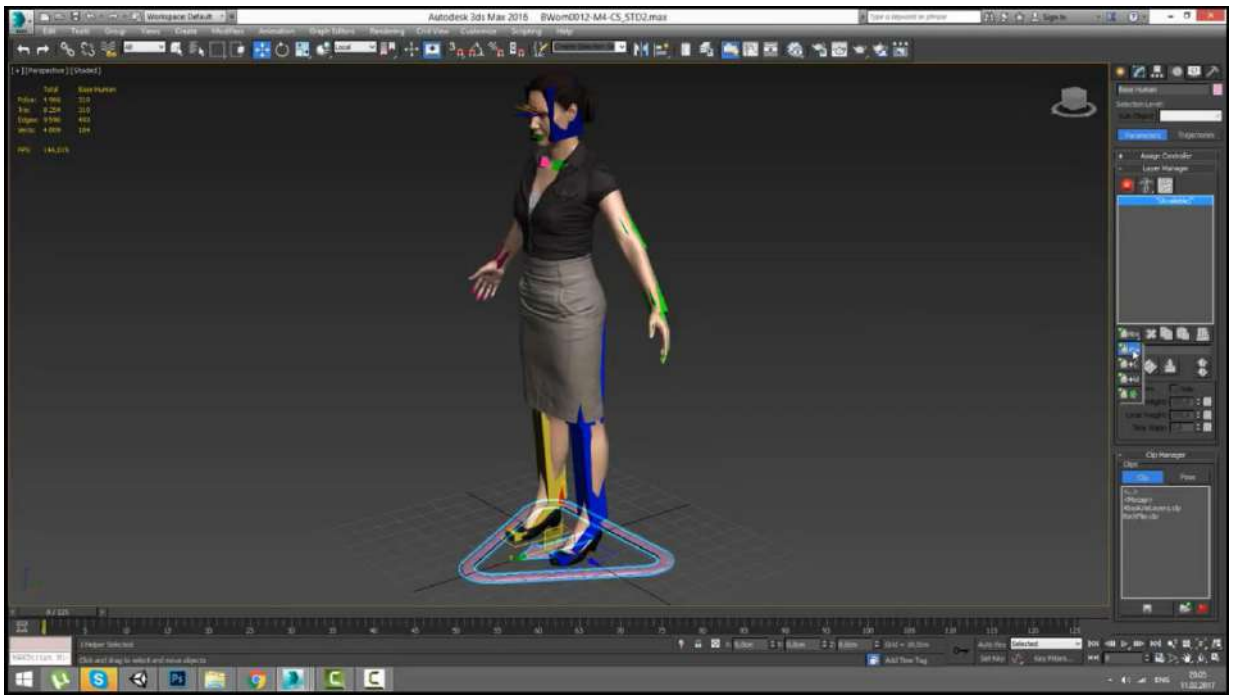


Рисунок 11.10 – Модель людини у стані спокою

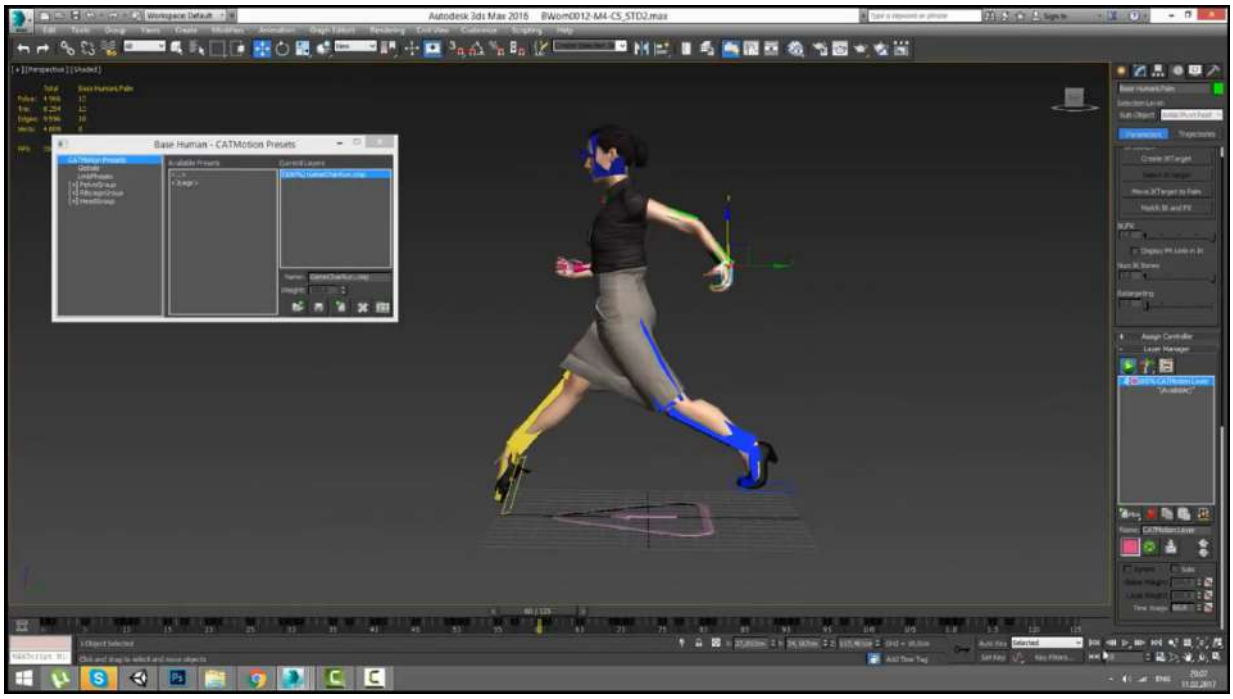


Рисунок 11.11 – Модель людини під час імітації бігу

Вимкнення відображення кісток скелета виконується у вкладці Display (рис. 11.10 – 11.11).

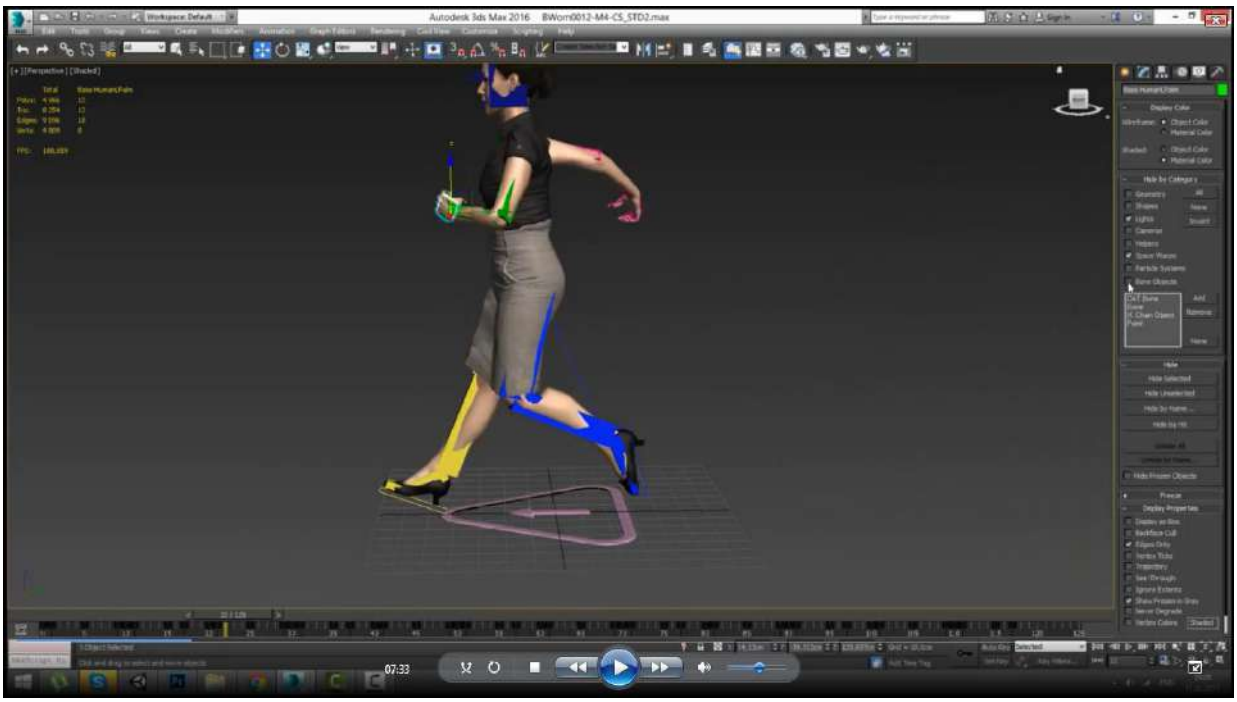


Рисунок 11.12 – Вигин суглобів

Як видно на рис. 11.12–11.15, при вигині суглобів виконується зайва деформація полігональної сітки моделі, що не відповідає природі людини. Це пов’язано з тим, що ті ж самі полігональні межі присвоєні згідно ваговому впливу декількох кісток, і виконується подвійна деформація.

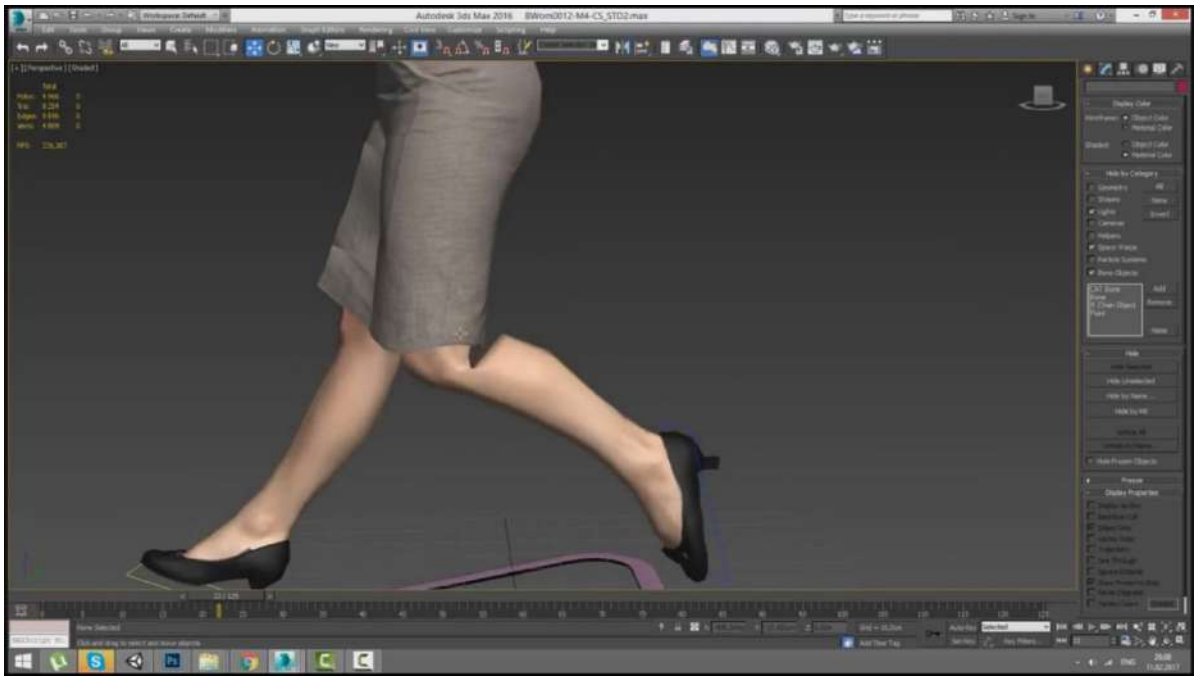


Рисунок 11.13 – Вигин суглобів і подальше викривлення полігональної сітки моделі

Рисунок 11.14 – Демонстрація неправильного викривлення поверхні моделі

Для того щоб не відбувалося таких заломів сітки (рис. 11.14), необхідно зайти в налаштування скінінга та вручну задати ступінь впливу кожної кістки на ту чи іншу область поверхні. У місцях вигину не повинно бути 100 %-ї ваги впливу від двох кісток, рекомендовано зменшити ступінь впливу кожної з сусідніх кісток до 50 %, або менше (рис. 11.15).

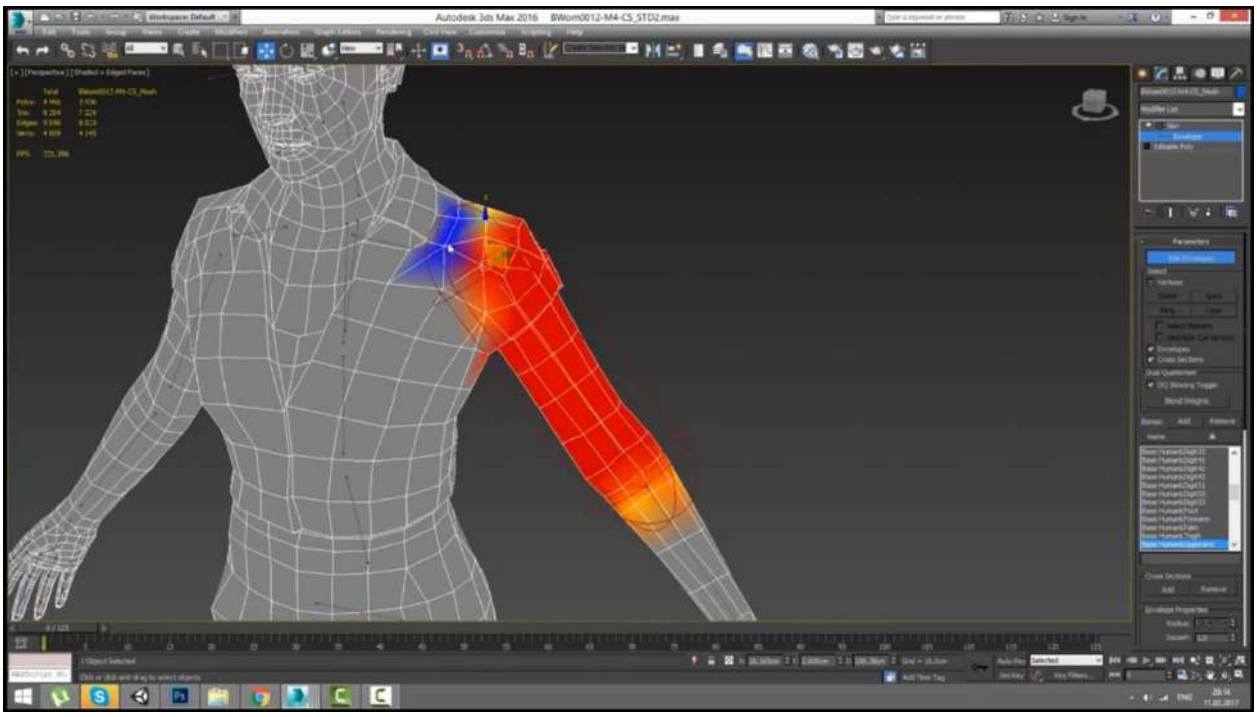


Рисунок 11.15 – Ступінь впливу прив’язки сітки до скелета моделі

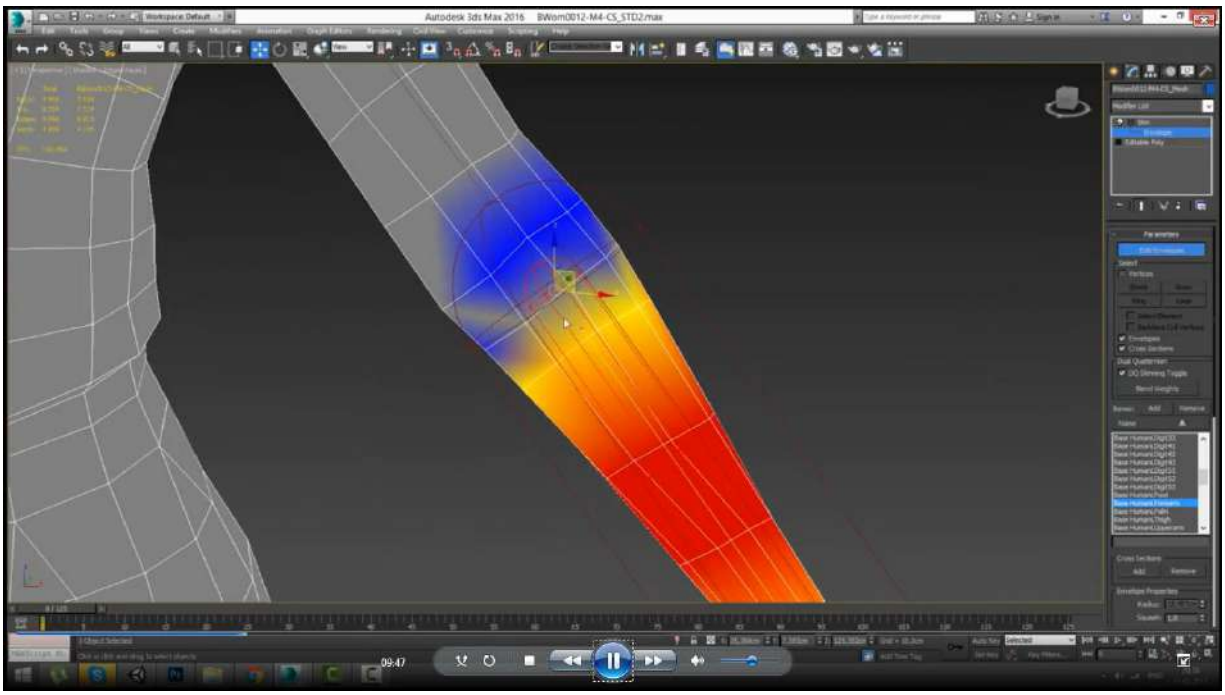


Рисунок 11.16 – Ступінь впливу прив’язки сітки до скелета моделі

Також автоматичний скінінг може давати збої, як на цьому скрині знизу, де на повороті голови зачіпається фрагмент плеча (рис. 11.17).

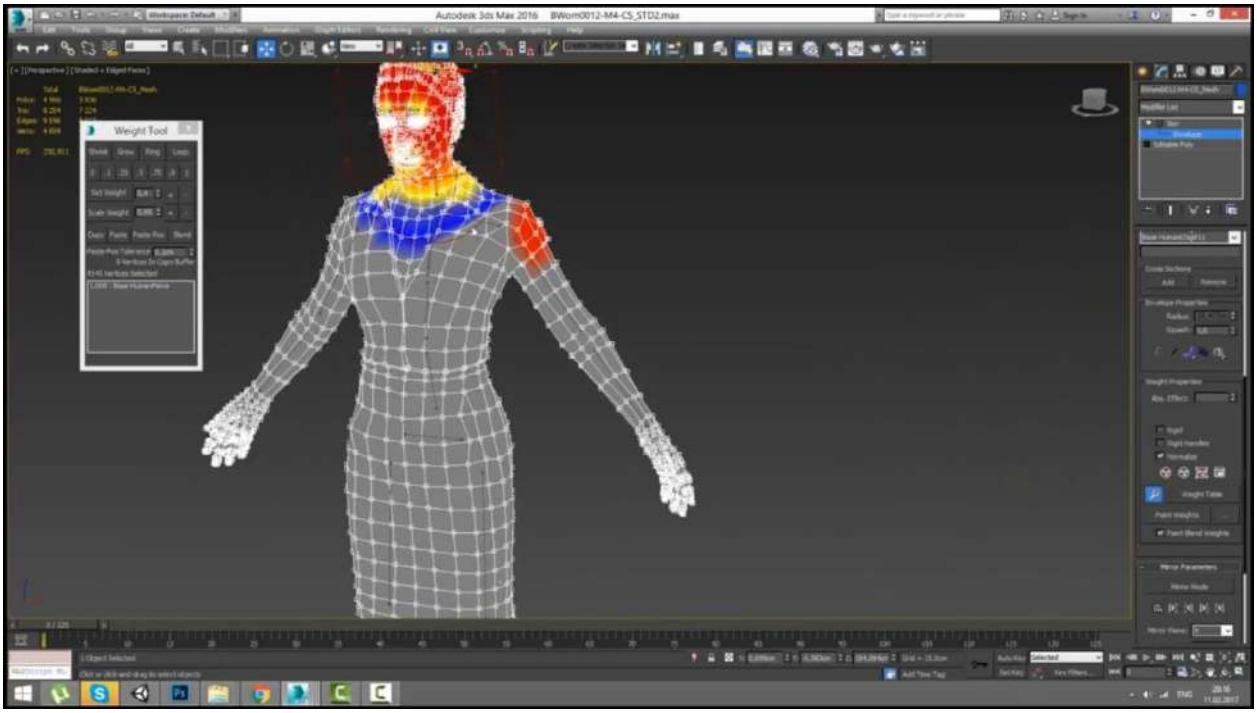


Рисунок 11.17 – Місця перетину кісток у суглобах

У налаштуваннях модифікатора **Skin** необхідно вибрати редагування вершин і перевірити кожну із задіяних кісток скелета. У місцях перетину кісток в суглобах знизити значення менше 50 % (рис. 11.16) і знову перевірити анімацію руху.

Окрім редагування ваг кожної кістки, слід перевірити правильність опорних точок кісток в місцях вигину моделі, і зрушити їх при потребі.

Наступним важливим кроком є редагування полігональної сітки моделі. Як видно з рис.11.18, сама сітка може некоректно розтягуватися, для цього в місцях сильного вигину потрібно збільшувати число полігонів або змінити їх форму, розрахувавши запас вигину (рис. 11.19).

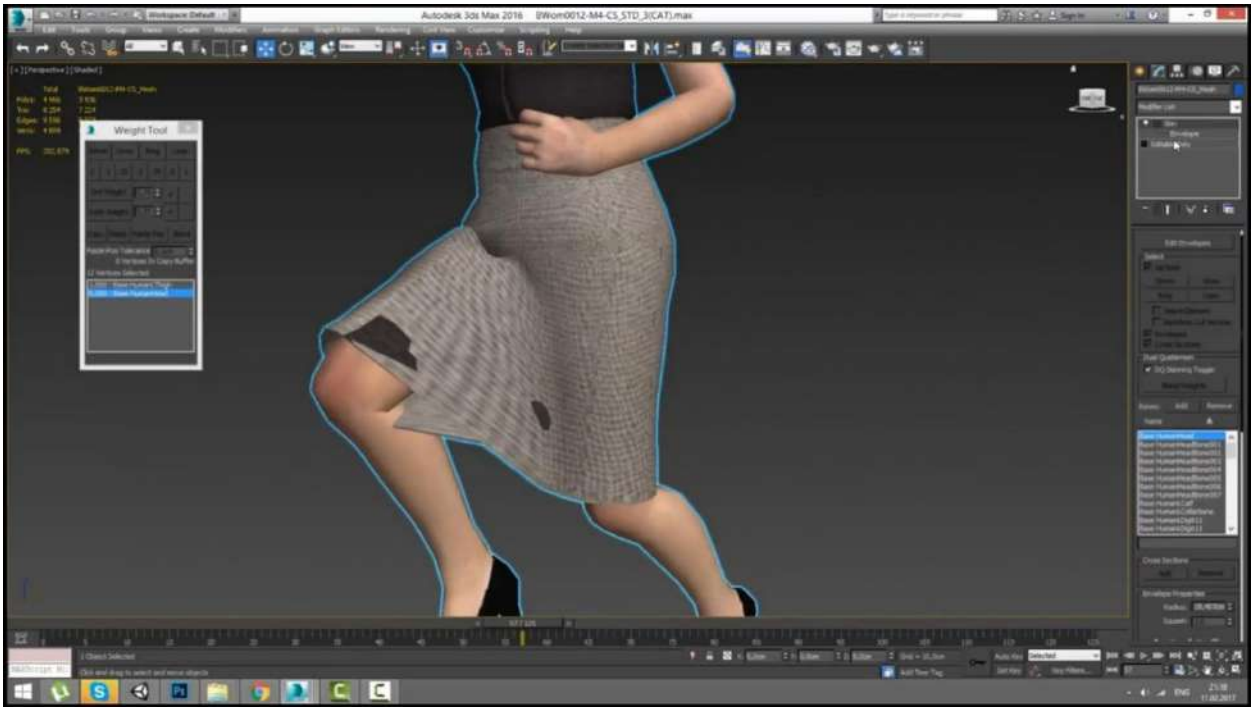


Рисунок 11.18 – Приклад некоректної деформації полігональної сітки моделі

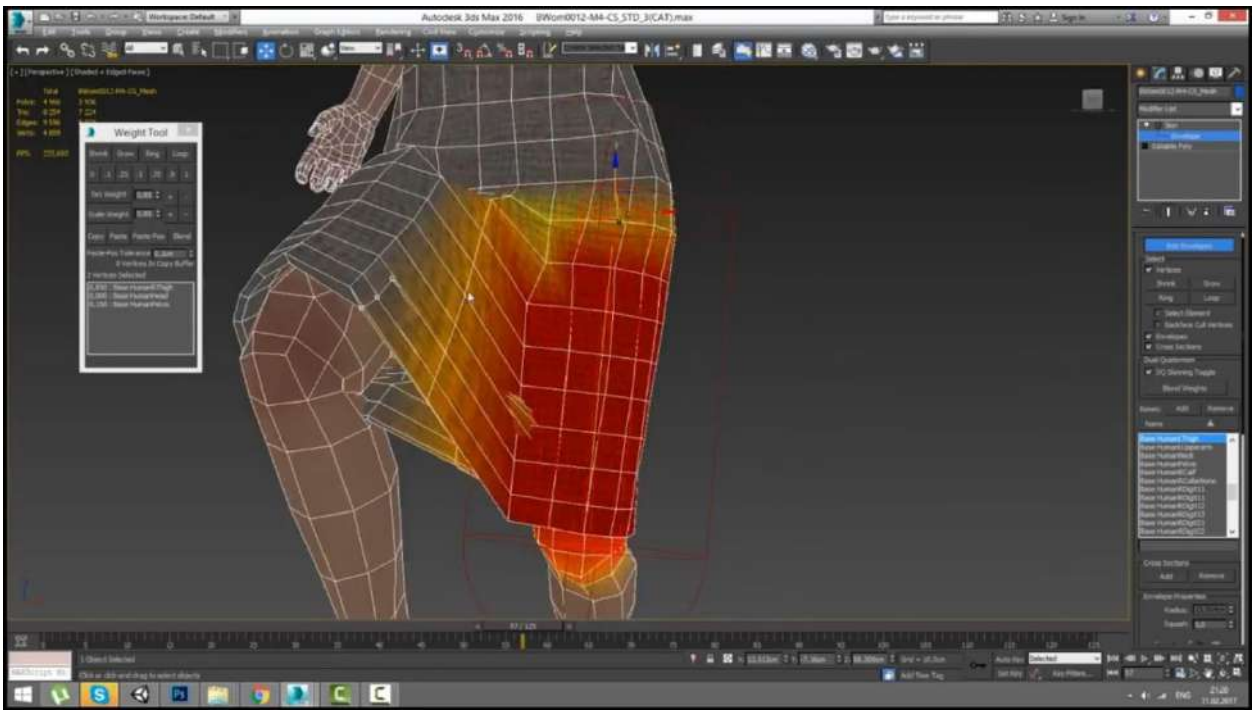


Рисунок 11.19 – Місця деформації полігональної сітки моделі

Внаслідок цього, виконавши всі перераховані маніпуляції, одержуємо необхідний результат (рис. 11.20-11.21).

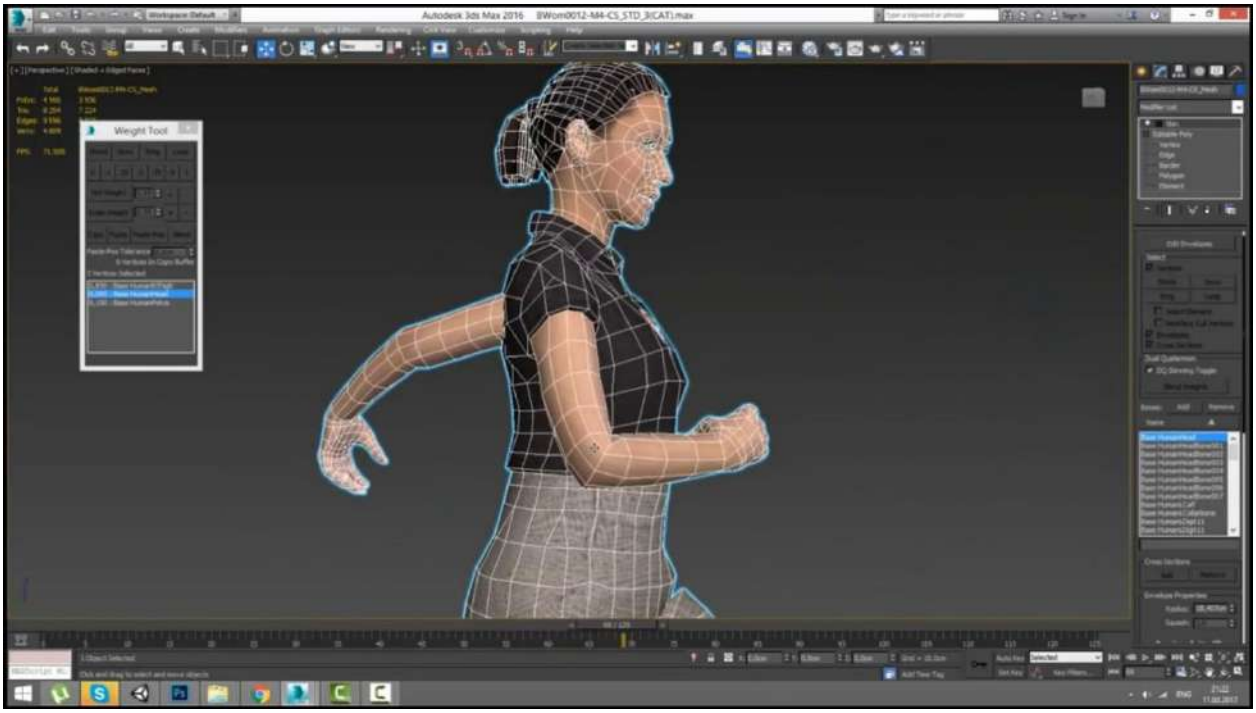


Рисунок 11.20 – Аналіз деформації полігональної сітки моделі

Рисунок 11.21 – Правильний рух полігональної сітки моделі під час її анімації

Наступним кроком в еволюції захоплення рухів стало запрошення іменитих зірок Голлівуду для гри міфічних персонажів, таких як Білл Найі (персонаж Дейві Джонса на рис. 11.22) і Енді Серкіз (Володар Перснів на рис. 11.23).

Рисунок 11.22 – Актор Білл Найі і його персонаж Дейві Джонс

Рисунок 11.23 – Актор Енді Серкіс та його персонаж Голлум

11.3. Фізика поведінки об’єктів

Для того щоб об’єкти, які пересуваються в сцені, взаємодіяли один з одним, виглядали правдоподібно і відповідали своїм реальним аналогам, симуляції форми поверхонь, освітлення і матеріалів мало. Необхідно також враховувати фізику поведінки об’єктів. Як відомо, кожен об’єкт має свою масу, твердість і шорсткість поверхні, що безпосередньо впливає на взаємодії з іншими фізичними об’єктами. Також на об’єкти, крім сили світла, можуть впливати інші сили природи: сила тяжіння, вітер, вибухи і т. д. Починаючи з версії 3DS MAX 7, з’явився плагін симуляції фізичних явищ **Reactor**, побудований на фізичному движку **Havok**. Пізніше в 2012 році, він був замінений на движок **MassFX** (рис. 11.24) від **NVIDIA**. Дані модифікатори дозволяють імітувати більшість фізичних явищ природи і властивостей матеріалів для спрощення створення ключових кадрів анімації об’єктів. Тепер не потрібно покадрово виставляти положення об’єктів, що взаємодіють між собою або рухаються під впливом гравітації.

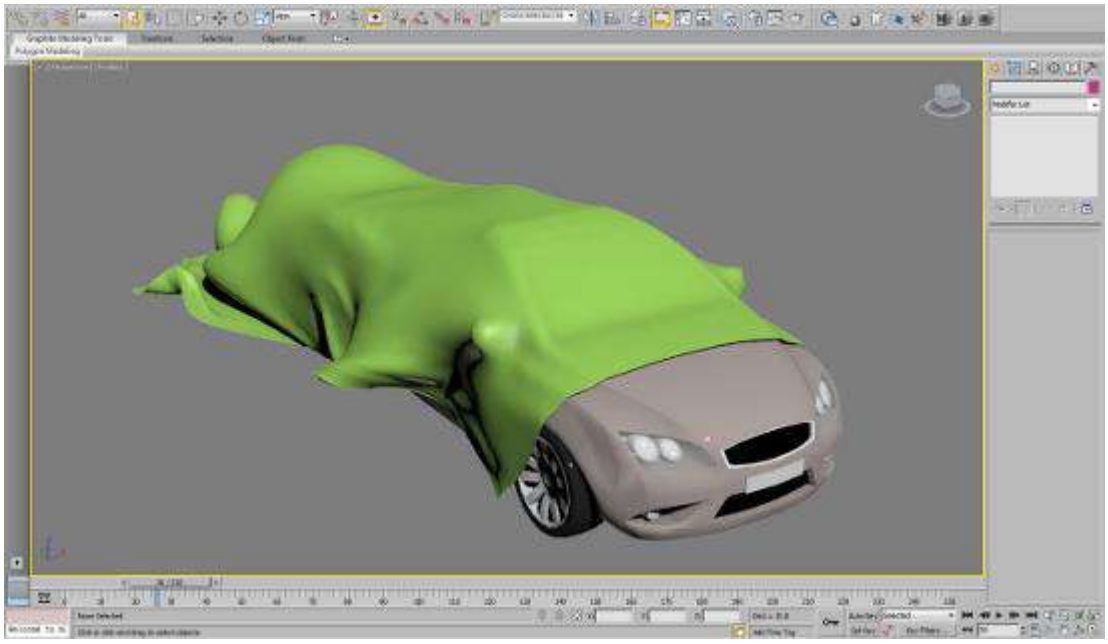


Рисунок 11.24 – Імітація фізики руху тканини в MassFX

11.3.1. MASSFX

В 3DS MAX представлена утиліта, що імітує вплив на об’єкти законів фізики (рис. 11.25), таких як сила тяжіння, сила тертя, сила вітру і т. д. Існує три типи об’єктів:

– **динамічні,** які рухаються при зіткненні з іншими об’єктами або під впливом гравітації і інших сил;

– **кінематичні –** анімуються стандартними методами, сили природи на них не впливають;

– **статичні –** абсолютно тверді елементи від яких будуть відскакувати динамічні об’єкти.

Додаткові налаштування плагіна:

– сила гравітації **(Global Gravity);**

– режим сну **(Sleep setting)** – величина в кількості кадрів нашої анімації, – при якій об’єкти знаходяться в статиці, тобто на них ніщо не впливає;

– швидкість зіткнення і швидкість стрибучості;

– **Contact Shell (Дистанція і глибина зіткнень).**

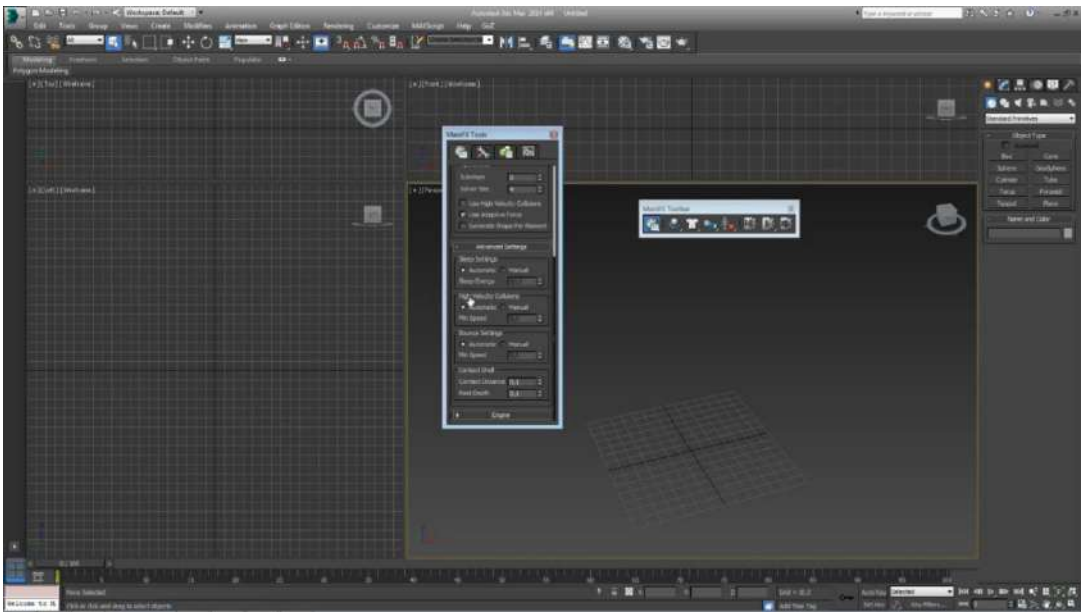


Рисунок 11.25 – Налаштування **Mass FX**

11.3.2. Налаштування маси, щільності, пружності і тертя об'єкта

Налаштування **Rigid Bodies Substeps** (рис. 11.26) – вказується кількість ступенів прорахунку фізики, за замовчуванням використовується мінімальне значення для прискорення прорахунку, але можна збільшити цю кількість.

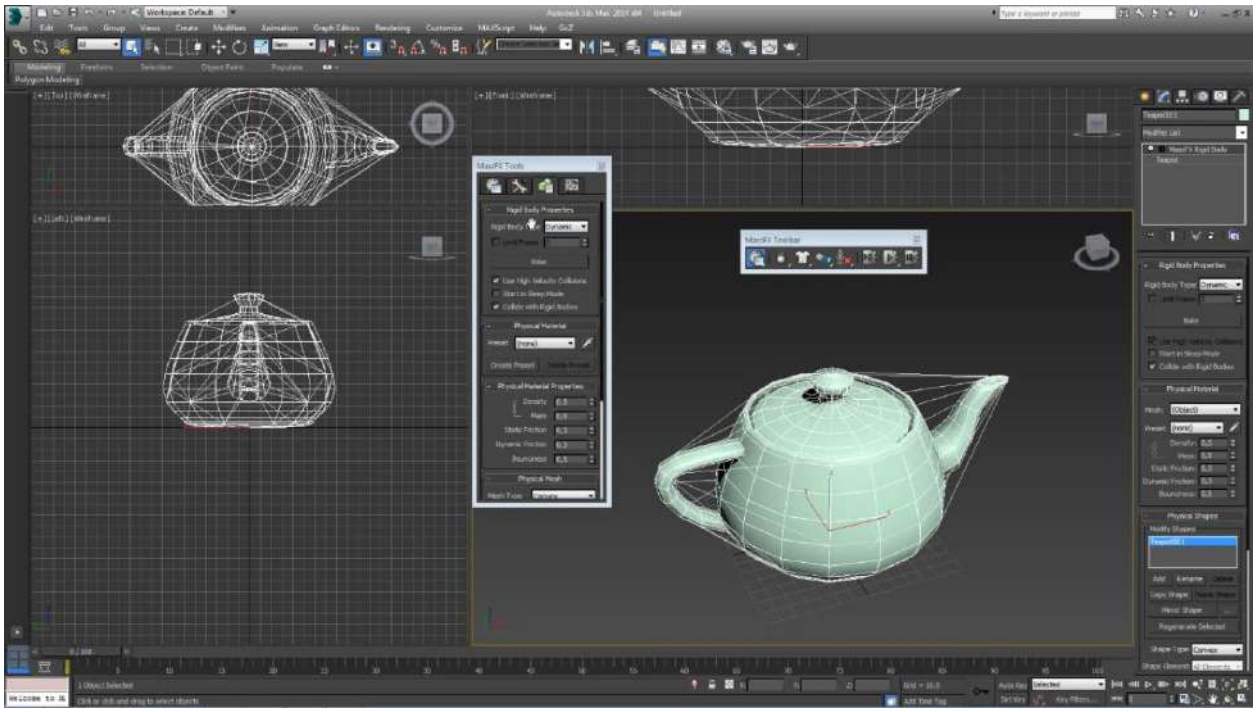


Рисунок 11.26 – Кількість ступенів прорахунку фізики

Фізика взаємодії не може розраховуватися безпосередньо з полігональної сіткою поверхні. Для спрощення і прискорення розрахунків **MassFX** самостіно генерує невидиму поверхню навколо об'єкта, за замовчуванням для базової моделі чайника вона становить всього 32 полігони. Цю величину, а також форму фізичної оболонки можна регулювати, збільшуючи і зменшуючи зону впливу (рис. 11.27).

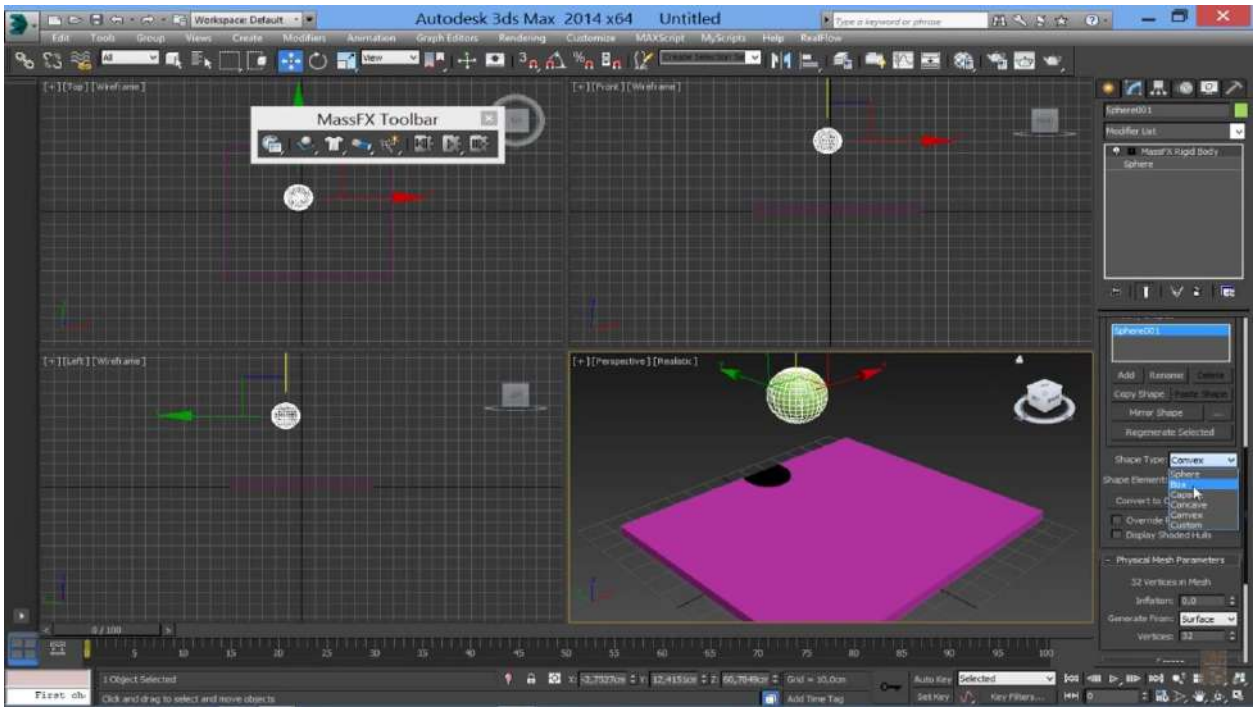


Рисунок 11.27 – Імітація фізики падіння кулі

Список імітації сил наведено в тулбарі налаштувань **MassFX** (рис. 11.28).

Рисунок 11.28 – Список імітації сил

Команда **Bake** дозволить «запекти» ключові кадри анімації, щоб з разу в раз отримана анімація рухалася вже по ключовим кадрам таймлайна.

Створення обмежувачів і ієрархічних ланцюжків. Ця функція **(Constraint)** дозволяє створювати **Dummy** об’єкти, що взаємодіють з іншими, тягнуть ієрархічний ланцюжок. Це необхідно, щоб модель, яка складається з декількох об’єктів, не розвалювалася під дією ваги, була цілісною, але в той же час кожен об’єкт моделі впливав на фізику руху по-своєму (рис. 11.29).

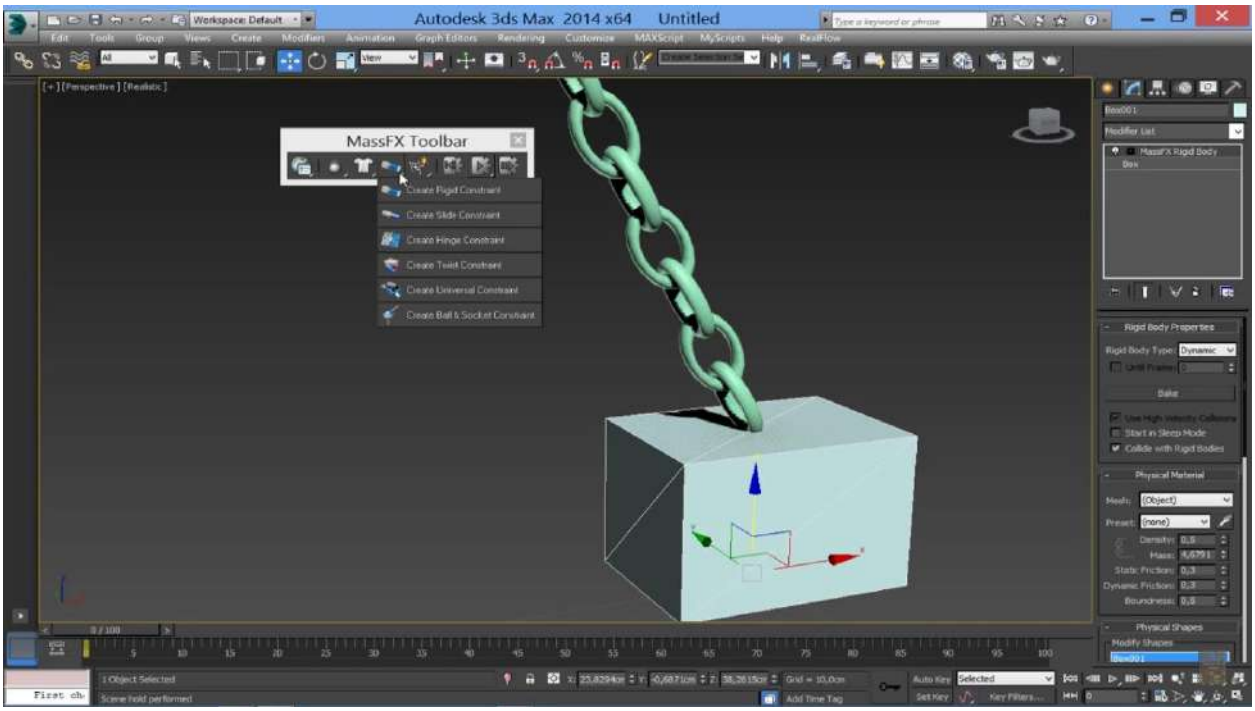


Рисунок 11.29 – Рух куба, на який діє сила тяжіння

Великий набір змінюваних параметрів дозволяє отримати матерію з необхідними характеристиками – від грубої шкіри до невагомого шовку (рис. 11.30).

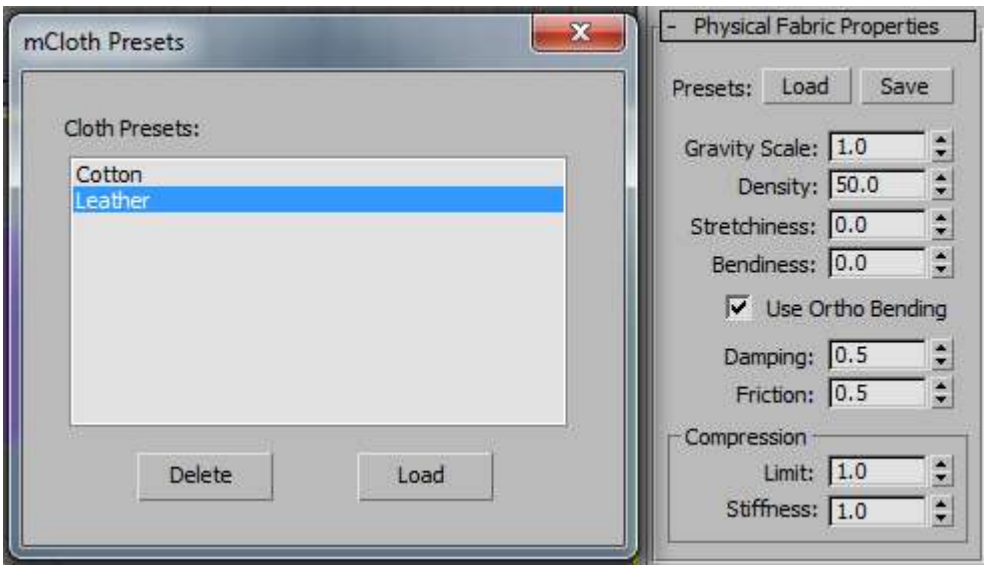


Рисунок 11.30 – Вибір параметрів матеріалу тканини

Додатково можна контролювати точність фізичних обчислень і силу взаємодії з твердими тілами. Команда **mCloth** (рис. 11.31) підтримує можливість розриву тканини під дією прикладеної сили.

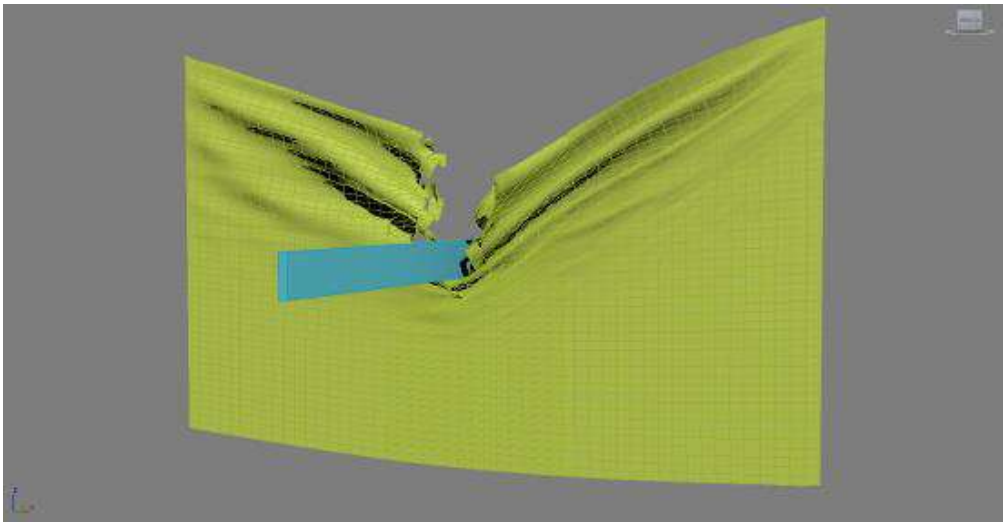


Рисунок 11.31 – Імітація розриву тканини **mCloth**

Однією з основних особливостей **MassFX** 2013 стала поява окремого модуля для розрахунку фізики тканини – **mCloth**. Другим модулем з прорахунків фізики твердих частинок є **mRigids**. Великий набір змінюваних параметрів дозволяє отримати матерію з необхідними характеристиками – від грубої шкіри до невагомого шовку – **mCloth** підтримує можливість розриву тканини під дією прикладеної сили. Також є опційнавізуалізація напружень в

матеріалі. До іншої важливої особливості **mCloth** слід віднести **"ballon behavior"** – замкнуті поверхні можуть бути наповнені внутрішнім тиском (рис. 11.32-11.33), за рахунок чого можна імітувати поведінки м'яких або пружних тіл (наприклад, тих же повітряних куль).

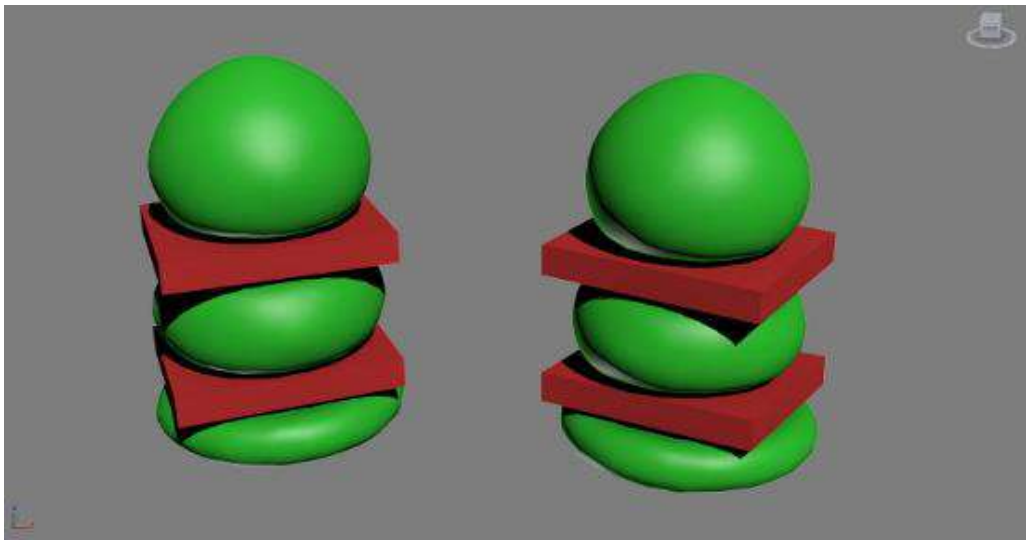


Рисунок 11.32 – Імітація тиску всередині пружних об'єктів

mCloth є рішенням для розрахунку фізики поведінки тканини, проте, не позбавлене ряду проблем як, наприклад, дуже низької швидкості запису в ключові кадри в порівнянні з симуляцією у вьюпортах.

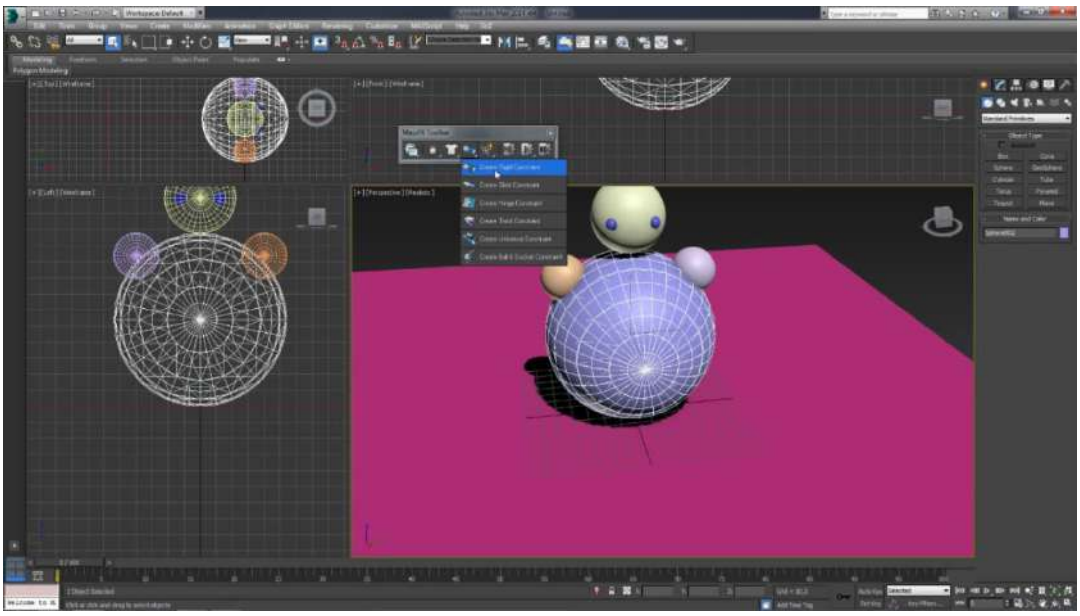


Рисунок 11.33 – Імітація круглих тіл і впливу на них сили тяжіння

11.1.3. Модифікатор руйнування об’єктів Ray Fire

3DS Max також дозволяє здійснювати руйнування складових об’єктів під дією різних сил, таких як сила тяжіння (рис. 11.34 – 11.35).

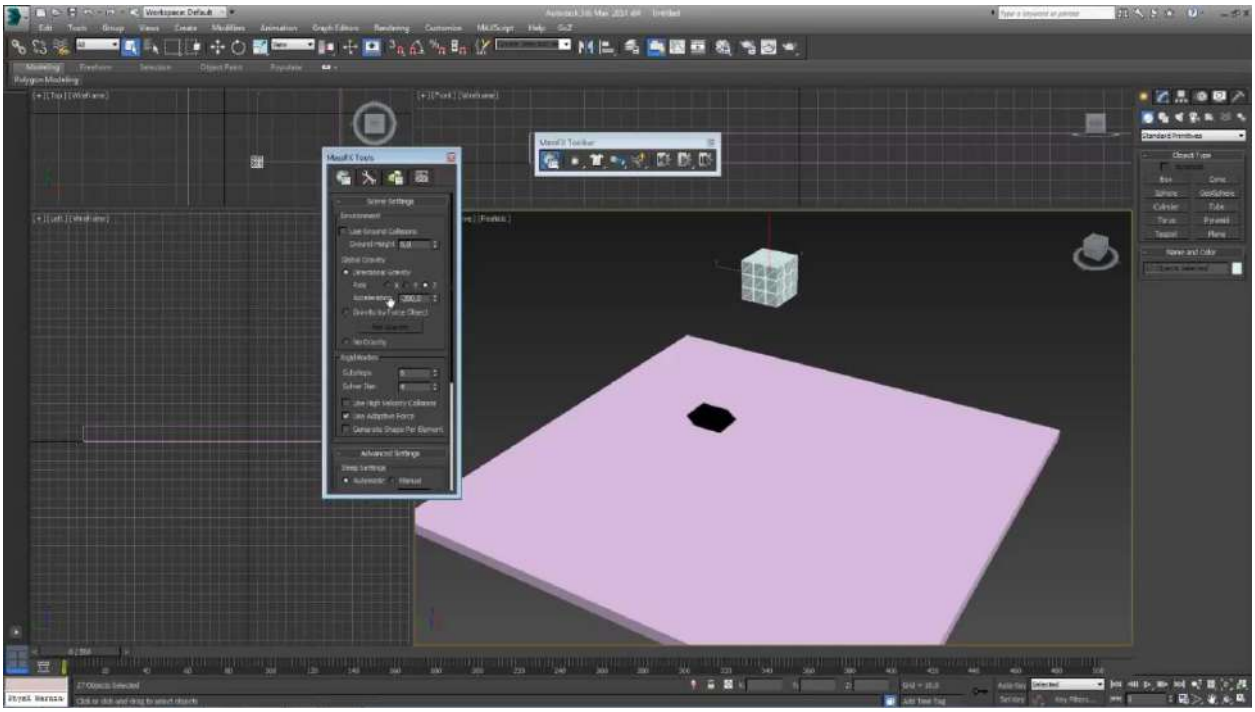


Рисунок 11.34 – Запуск імітації падіння складового об’єкта

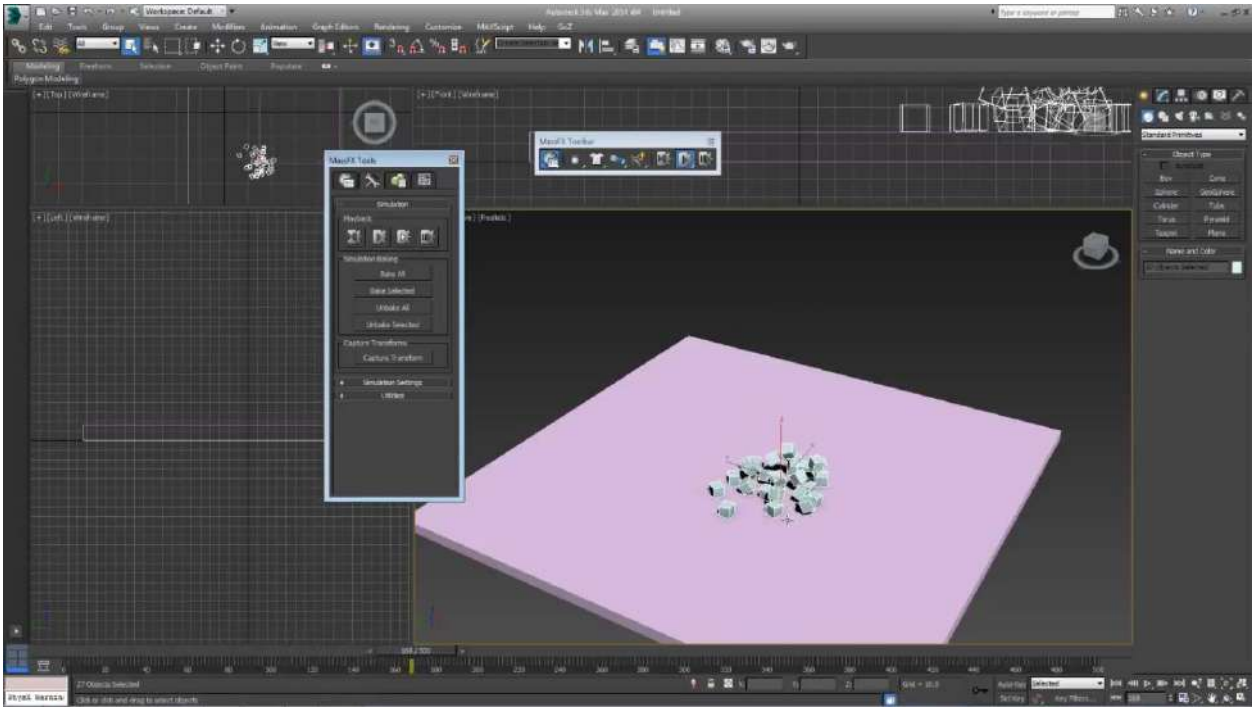


Рисунок 11.35 – Руйнування складового об’єкта при падінні на тверду поверхню

Плагін **Ray Fire** знаходиться у вкладці примітивів і дозволяє імітувати взаємодію твердих об’єктів. Для цього необхідно у вікні **Dynamic Objects** (рис. 11.36) додати об’єкти зі сцени, які будуть руйнуватися, а у вікні **Static Pbjects** додати ті об’єкти, які взаємодіють з нашим динамічним об’єктом.

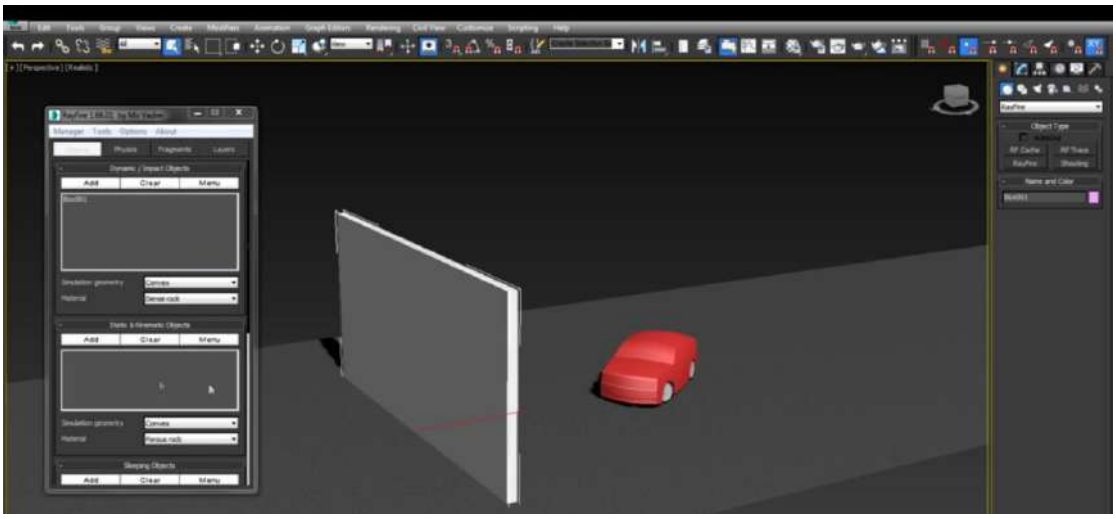


Рисунок 3.36 – Плагін **Ray Fire**

Далі у вкладці **Frament** необхідно додати кількість фрагментів руйнування об’єкта (рис. 11.37). Наприклад 300 фрагментів (рис. 11.38).

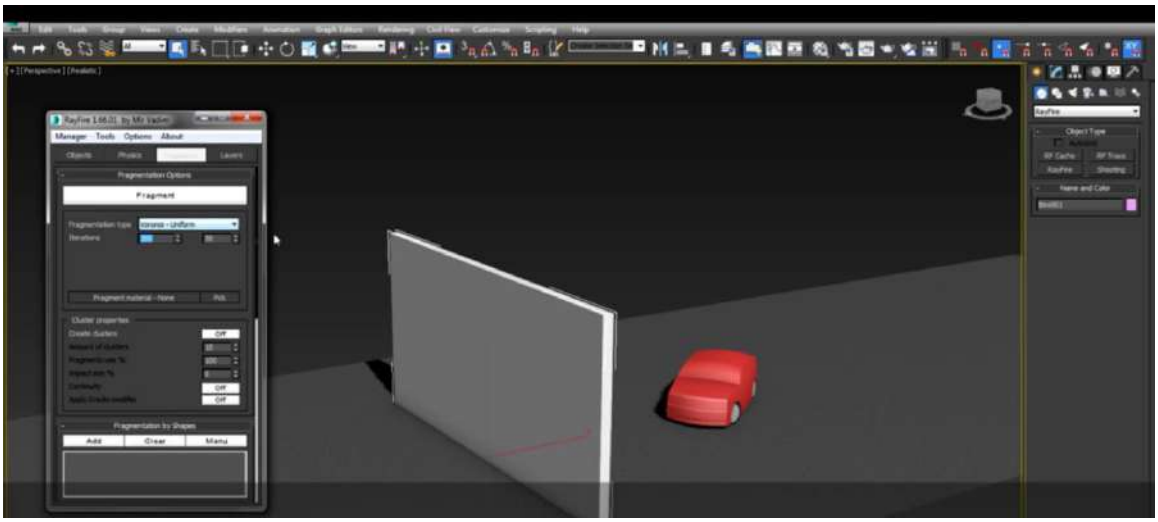


Рисунок 11.37– Імітація стінки з 300 фрагментів

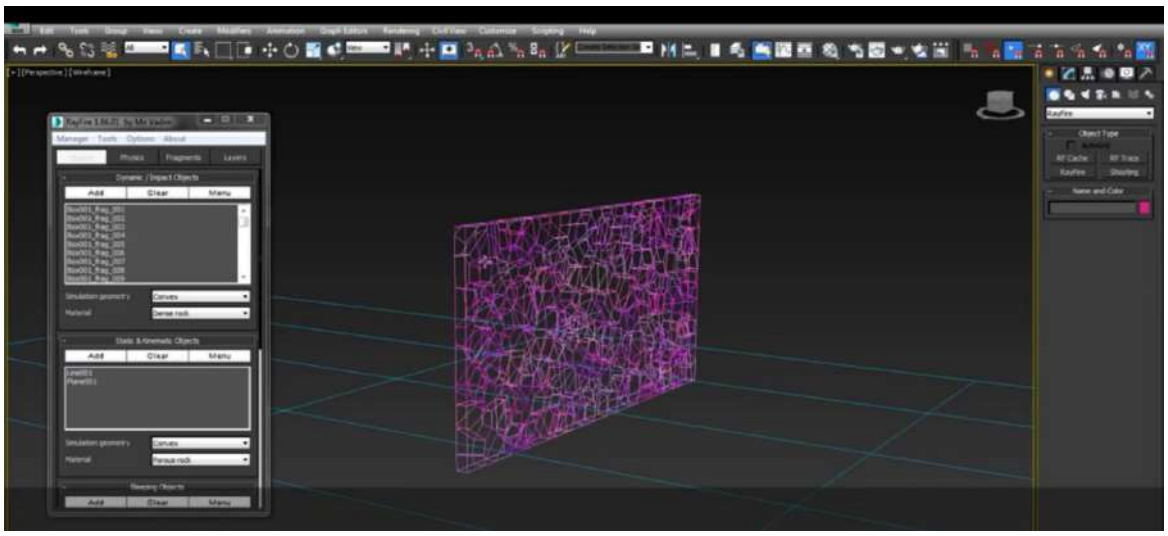


Рисунок 11.38 – Імітація руйнування стіни із заданого числа фрагментів

Далі кнопка **Menu - Send to sleeping list** дозволяє додати всі новоутворені 300 фрагментів конструкції (рис. 11.39). Ці об’єкти будуть в режимі очікування до тих пір, поки з об’єктом не зіткнеться інший **Static objects**.

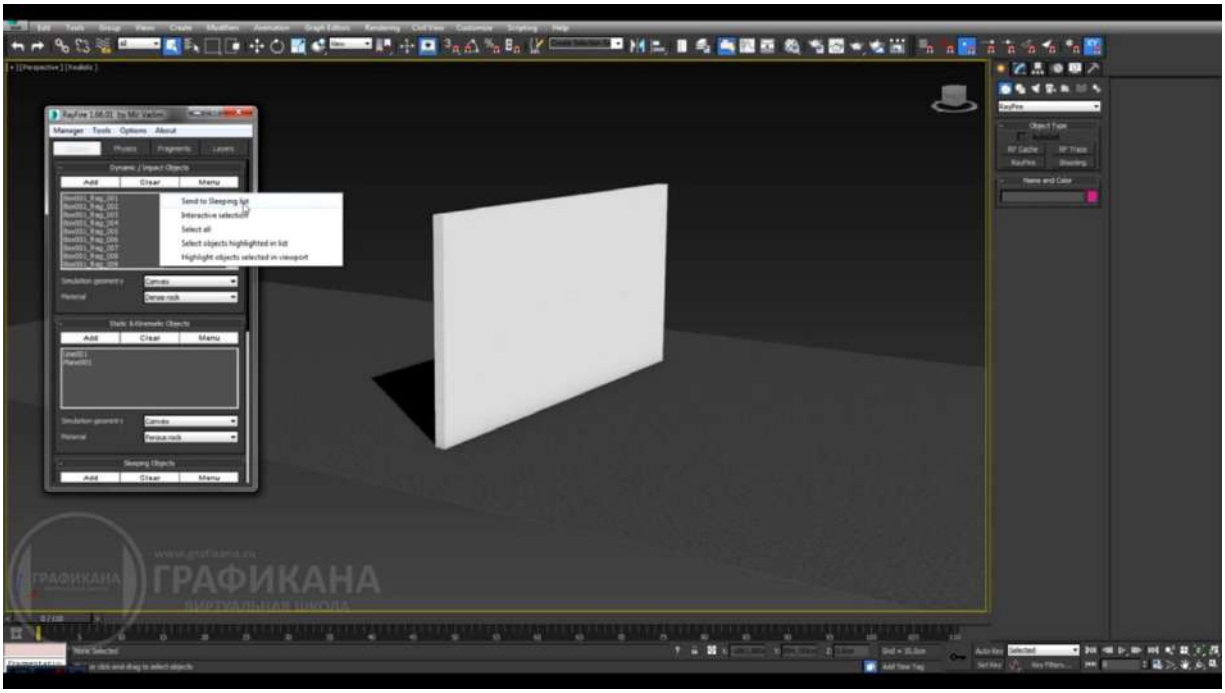


Рисунок 11.39 – Режими очікування конструкції

Команда **Bake** відповідає за «запікання» анімації, тобто прорахунок взаємодії і руйнування об’єктів (рис. 11.40).

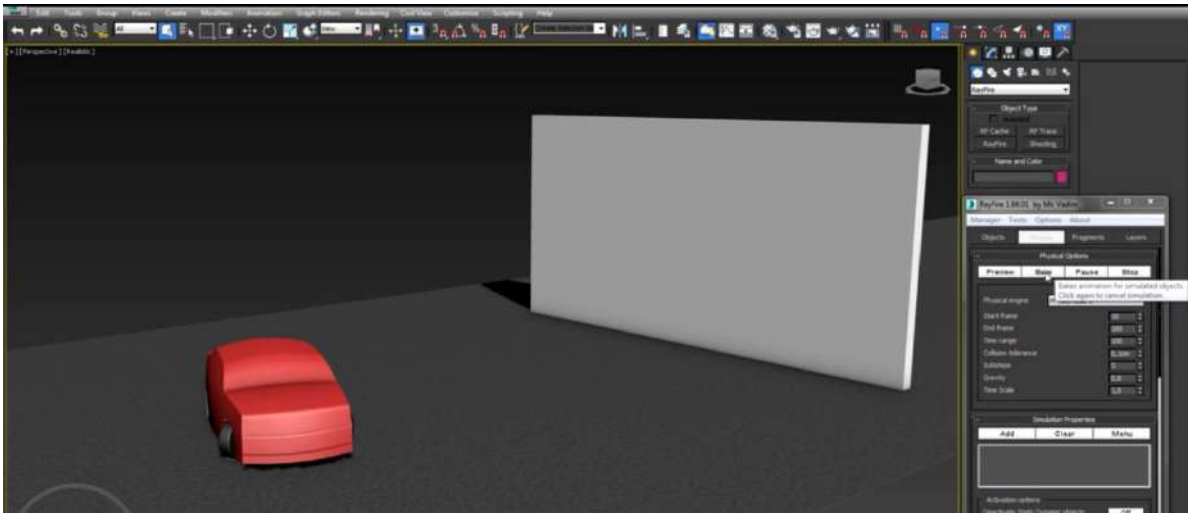


Рисунок 11.40 – Прорахунок взаємодії і руйнування об’єктів

На рис 11.41 поданий результат виконання анімації руйнування стінки з 300 віртуальних фрагментів від зіткнення з твердим тілом (модель авто).

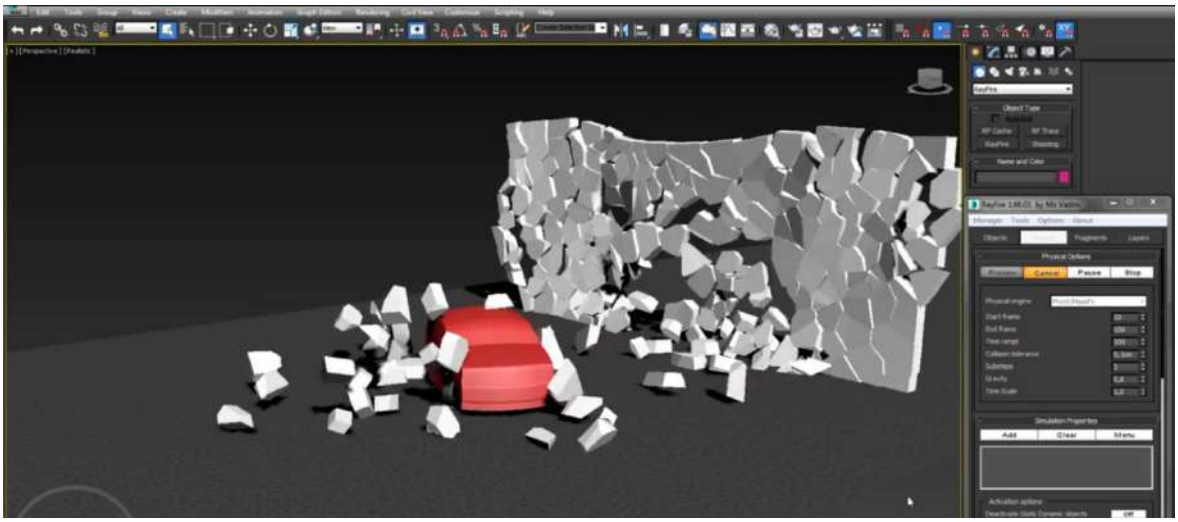


Рисунок 1.41 – Анімація руйнування стіни з 300 фрагментів від зіткнення з твердим об’єктом

У версіях 3DS MAX після 2018 року з’явилися нові функціонали, такі як Об’ємні деформації (**Space Warps**) – Динамічні сили (**Forces**), **PBomb**, **Vortex** або **Wind**, вони отримали здатність впливати на **MassFX** об’єкти, як на тверді тіла, так і на тканину (рис. 11.42).

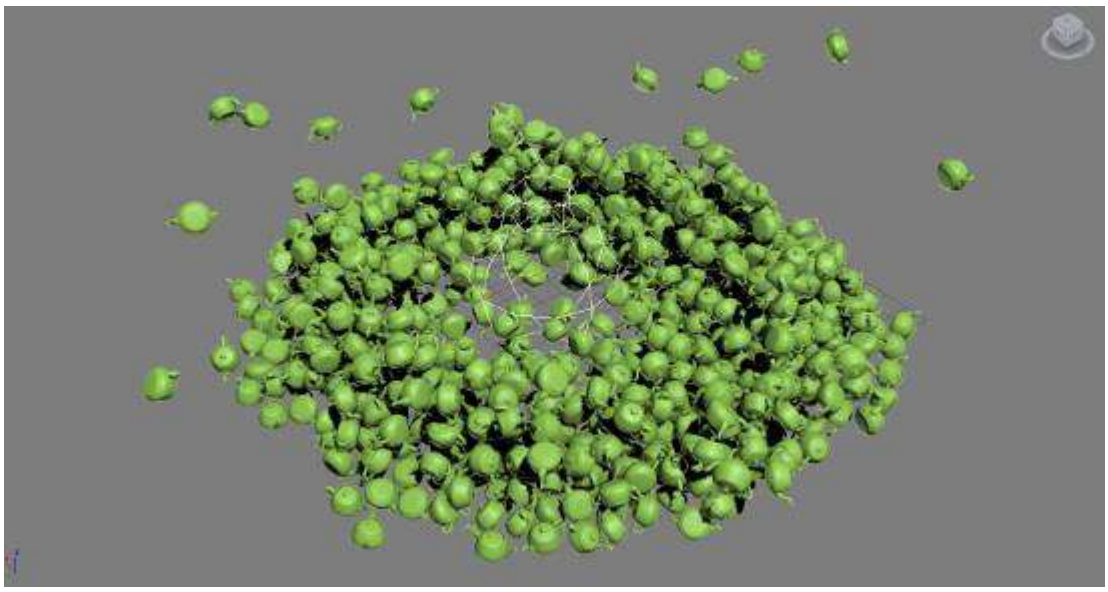


Рисунок 11.42 – Об’ємні деформації **(Space Warps)**

Даний механізм був розроблений в Autodesk і не використовує стандартні "силові поля" (**force fields**) зі складу **PhysX SDK**.

Додалися авто-генеровані динамічні рагдолли з biped скелетів (рис. 11.43).

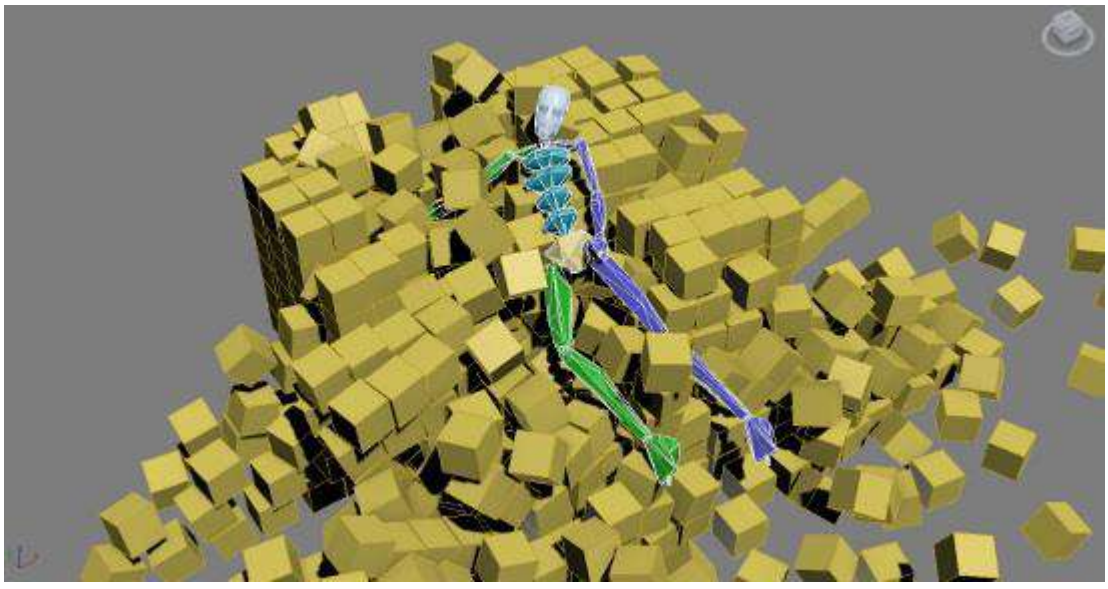


Рисунок 1.43 – Авто-генеровані динамічні об’єкти

Завдяки новому алгоритму, динамічні рагдолли тепер поводяться досить адекватно навіть без додаткового налаштування, а не скручуються дугою через кілька секунд після початку симуляції (як в попередніх плагінах).

Крім основних змін, описаних вище, в **MassFX 2013** присутня велика кількість дрібних поліпшень, виправлень і незначних оновлень інтерфейсу.

Розширено можливості з управління силою і напрямком гравітації в сцені (рис. 11.44).

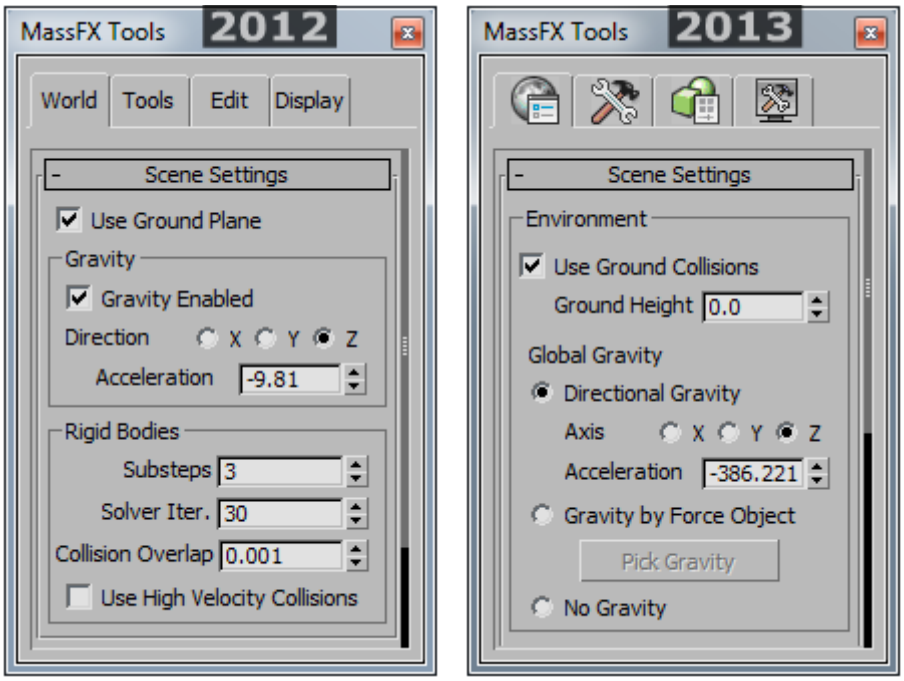


Рисунок 11.44 – Управління гравітацією в сцені

Алгоритм генерації опукло-ввігнутих об’єктів **(convex)** був поліпшений, а його інтерфейс став більш дружнім (рис. 11.45).

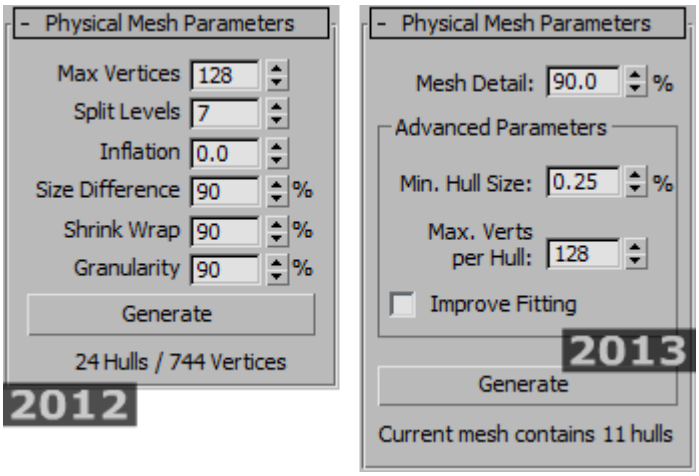


Рисунок 11.45 – Алгоритм генерації **Convex**

Фізичні меші тепер можна відображати не тільки у вигляді **wireframe** каркаса, але і поверхні із затемненням (**shaded hull**). Зручно для композитних об’єктів (рис. 11.40).

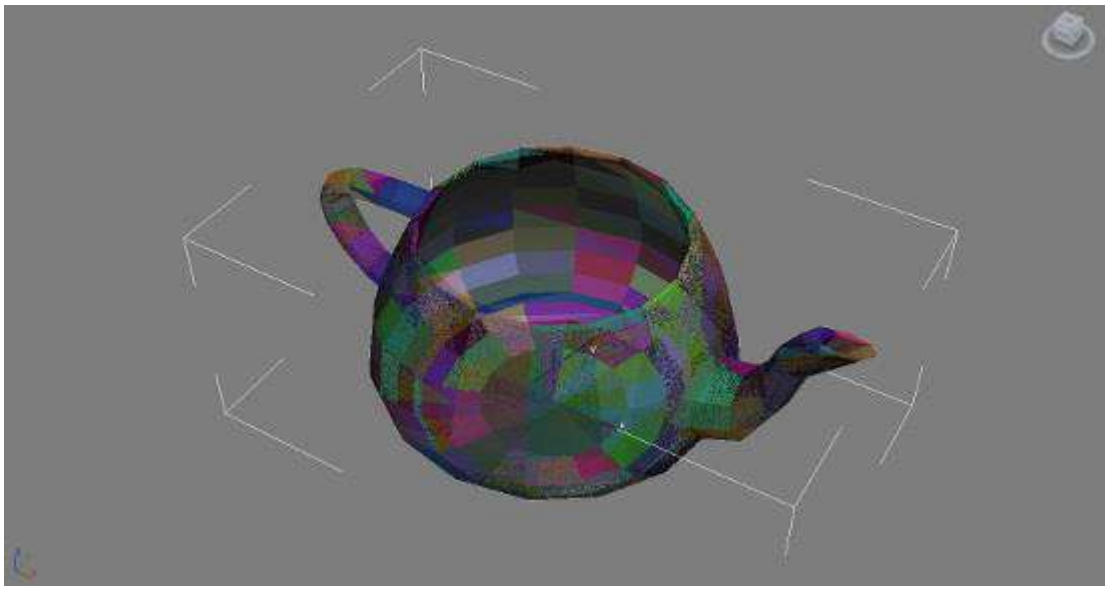


Рисунок 11.46 – Варіант відображення полігональної сітки моделі

На відміну від **Reactor**, імітація фізики моделі виконується для віртуальної спрощеної моделі, яка геометрично відрізняється від полігональної моделі 3DS MAX. Це зроблено з метою спрощення і прискорення обчислень, однак може створити труднощі при моделюванні фізично достовірно точних явищ. Тому модуль **Reactor** досі застосовують для моделювання точних розрахунків.

Ще одним додатковим модулем анімації фізики об’єктів є **Particle Flow**. За допомогою цього модуля можна створити практично будь-який ефект, пов’язаний з частками – бризки води, розбивання об’єкта на дрібні частини, іскри, що летять, пил і т. д.

11.4. Motion Capture (Захоплення руху)

Motion Capture – метод анімації персонажів і об’єктів, який знайшов широке розповсюдження в створенні сучасних комп’ютерних ігор і кінокартин. Також в індустрії його іноді називають скорочено – Мокап.

Суть цієї технології полягає в тому, що тіло реальної людини обвішують датчиками, кожен з яких відповідає вузлу руху в скелеті майбутньої тривимірної моделі персонажа. Фіксується рух актора за допомогою відео

камер або інфрачервоних датчиків і записується в спеціальний анімаційний трек.

Після цього траєкторія руху датчиків «прив'язується» до відповідних точок на «кістяк» тривимірної моделі.

Перші спроби з захопленням руху акторів були зроблені в кінофільмах «Термінатор 2 Судний день» і «Парк Юрського періоду» (1993 рік).

Студія **ALM** створила софт **Alia,s** за допомогою якого було створено 7965 кадрів тривимірної анімації для персонажа Т-1000. (рис. 11.47 – 11.50).

Рисунок 11.47 – Анімація персонажа Т-1000 (епізод 1)

Рисунок 11.48 – Анімація персонажа Т-1000 (епізод 2)

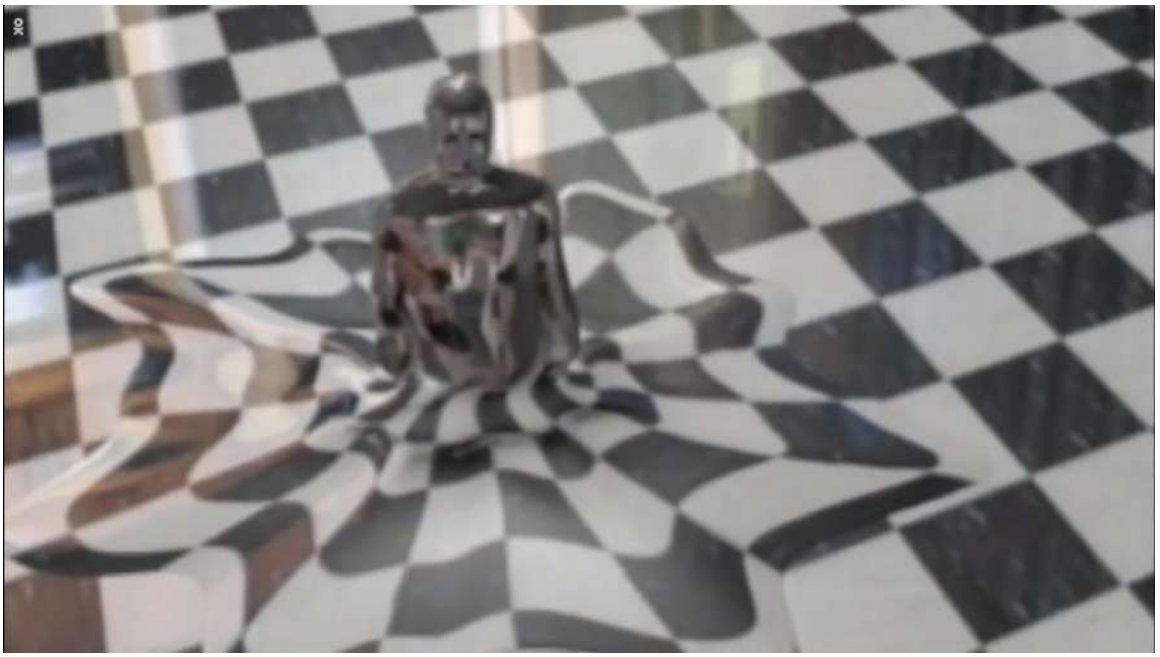


Рисунок 11.49 – Анімація персонажа Т-1000 (епізод 3)

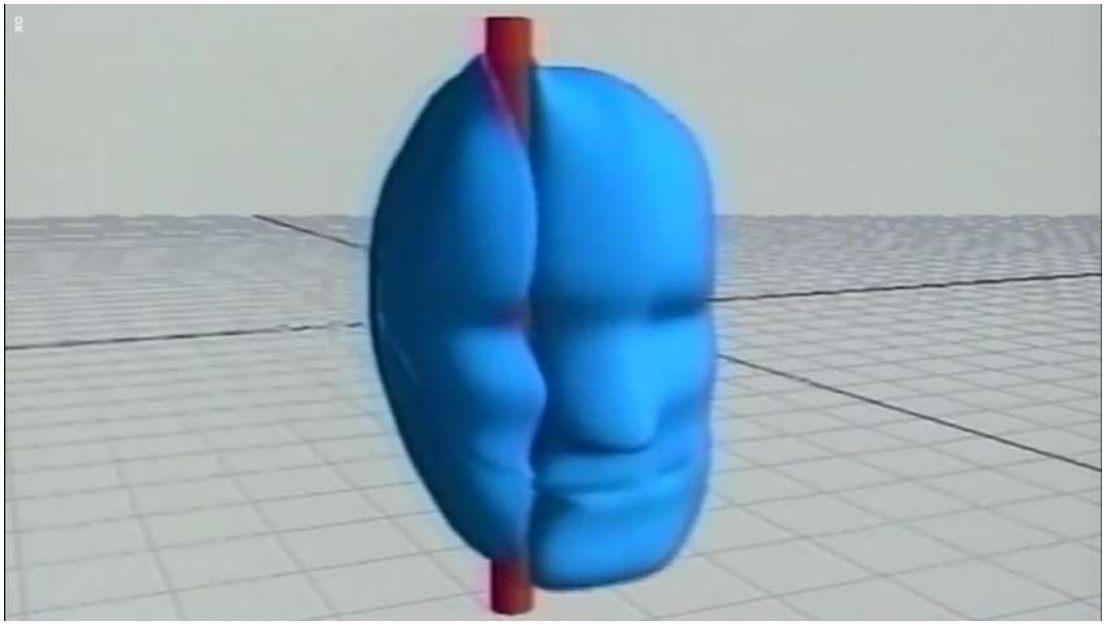


Рисунок 11.50 – Анімація персонажа Т-1000 (епізод 4)

Використовувалися фотографії актора з намальованими лініями на обличчі, щоб в подальшому по кривизні поверхні відтворити модель героя (рис. 11.5 – 11.52).

Рисунок 11.51 – Захоплення руху тіла актора (I)

Рисунок 11.52 – Захоплення руху тіла актора (II)

Також захоплення руху було вперше здійснено не тільки з тіла, але і з обличчя персонажа (рис. 11.53). Для відтворення фрази **Get Out** було багаторазово відскановано (рис. 11.54) обличчя актора і покадрово перенесено вираз міміки при озвучуванні кожної літери.

Рисунок 11.53 – Захоплення руху обличчя персонажа

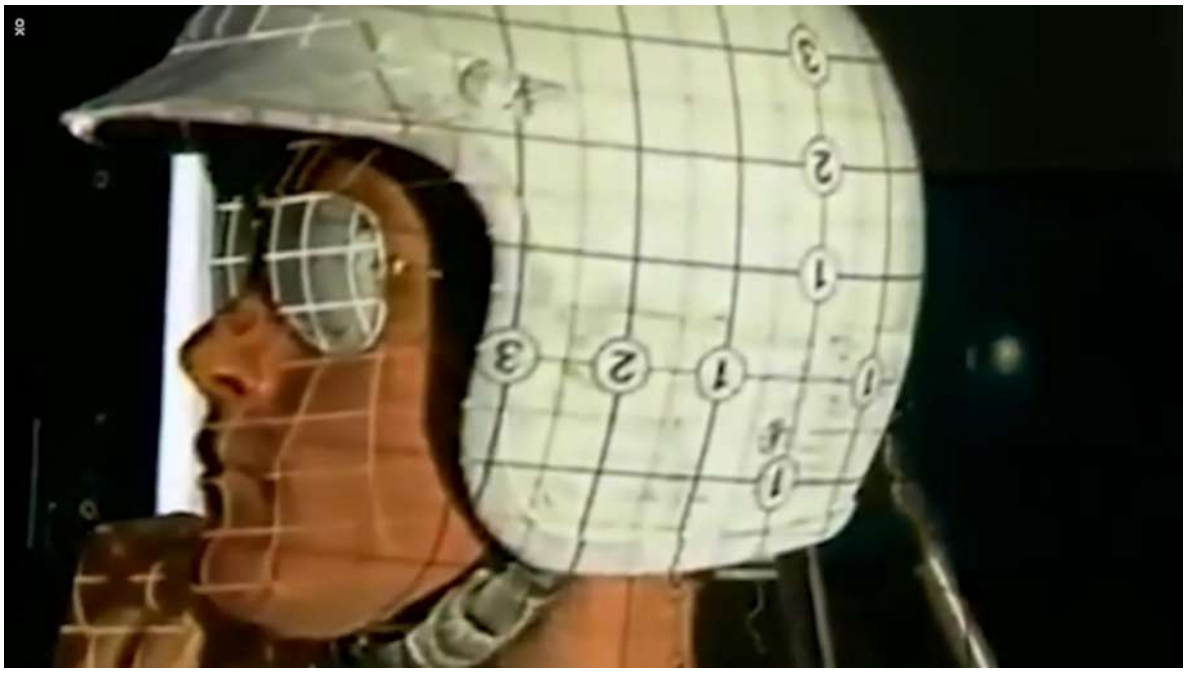


Рисунок 11.54 – Приклад багаторазово відсканованого обличчя актора при захопленні руху

Процес оцифровування і рендеру отриманих зображень здійснювався за допомогою **RenderMan** від **Pixar**, а відрисовка персонажів в першій бета версії **Photoshop** – від **Adobe**. На роботу 8000 кадрів пішло 8 місяців роботи 40 програмістів і аніматорів і 5,5 мільйона доларів (рис. 11.55 – 11.60).

Рисунок 11.55 – Процес оцифровування отриманих даних

Рисунок 11.60 – Програма **RenderMan** від **Pixar**

Для керування моделлю динозавра створили спеціальний джойстик під назвою **Digital Input Device** (**DID** – цифровий пристрій введення). Аніматори жартівливо називали його **Dinosaur Input Device** (пристрій введення динозаврів). Тут був задіяний джойстик, за допомогою якого був анімований рух робота-динозавра, тобто в реальному часі модель динозавра повторювала рухи рук аніматора (рис. 11.61).

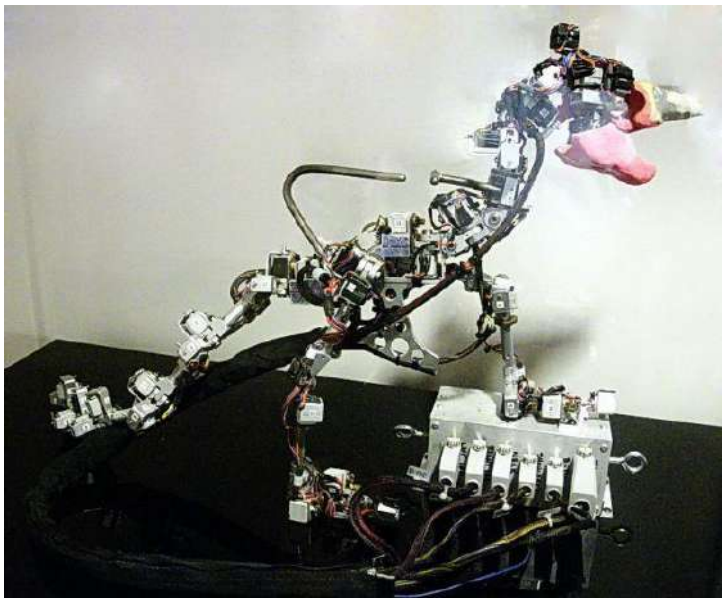


Рисунок 11.61 – Джойстик, за допомогою якого анімований рух робота-динозавра

Перший цифровий герой з’явився у фільмі 1999 року «Зоряні війни: Епізод 1 – Прихована загроза»; персонаж Джа-Джа Бінкс. Розробником є компанія Digital District (рис. 11.62).

Рисунок 11.62 – Персонаж Джа-Джа Бінкс

Перші фільми, створені цим методом, – це персонаж Голлум в трилогії Володар Перснів (2002 рік, персонаж Енді Серкіс). Це перша вдала спроба захоплення не тільки руху персонажа, але і міміки актора (рис. 11.63).

Рисунок 11.63 – Персонаж Голлум в трилогії Володар Перснів

Полярний експрес (2004 рік), перша спроба перенесення зовнішності і гри реального актора на анімаційного персонажа (рис. 11.64).

Рисунок 11.64 – Том Хенкс грає персонажа мультфільму Полярний Експрес

Я – робот (2004) – тільки примітивне захоплення руху декількох дублерів і заміна дублерів комп'ютерним персонажем.

Пірати Карибського моря («На краю світу» 2007 рік). Створення декількох персонажів з повним захопленням і міміки, і руху.

Аватар (2009 рік) – перша 3D віртуальна камера для оператора.

Повстання планети мавп (2011 рік) – перша взаємодія віртуального персонажа з реальними акторами в реальних знімальних декораціях.

Наступним кроком в еволюції захоплення руху стало запрошення іменитих зірок Голлівуду для гри міфічних персонажів, таких як Білл Найі (персонаж Дейві Джонса) і Вільям Дефо (Джон Картер) (рис. 11.65).

Рисунок 11.65 – Білл Найі і його персонаж Дейві Джонс з трилогії Пірати Карибського моря

Також наступним кроком стало запрошення все тих же акторів для гри анімаційних персонажів в комп'ютерних іграх, наприклад, Beyond Two Souls з тим же Вільямом Дефо в головній ролі (рис. 11.66).

Рисунок 11.66 – Вільям Дефо під час зйомок кінофільму Джон Картер (зліва) і гри **Beyond Two Souls**

На сьогоднішній день існує велика кількість маркерних систем захоплення руху. Різниця між ними полягає в принципі передачі руху:

1. **Оптично пасивні.** На костюмі, що входить в комплект такої системи, прикріплені датчики-маркери, які названі пасивними, бо відображають лише послане на них світло, але не світяться. У таких системах світло (інфрачервоне)

на маркери надсилається з установлених на камерах високочастотних стробоскопів і, відбившись від маркерів, потрапляє назад в об'єктив камери, повідомляючи тим самим позицію маркера (рис. 11.67 – 11.68).

Рисунок 11.67 – Датчики-маркери на акторах, що грають віртуальних кіноперсонажів

Рисунок 11.68 – Датчики, що зчитують розташування датчиків-маркерів

Мінус оптично пасивних систем полягає у тривалості розміщення маркерів на акторі. Так само іноді при швидкому русі або близькому

розташуванні маркерів один до одного система може їх плутати (оскільки ця технологія не передбачає ідентифікації кожного маркера окремо).

2. **Оптично активні** названі так тому, що замість світловідбивних маркерів, які кріпляться до костюма актора, в них використовуються світлодіоди з інтегрованими процесорами і радіосинхронізацією. Кожному світлодіоду призначається ID (ідентифікатор), що дозволяє системі не плутати маркери один з одним, а також впізнавати їх після того, як вони були перекриті і знову з'явилися в поле зору камер (рис. 11.69). У всьому іншому принцип роботи таких систем схожий з пасивними системами.

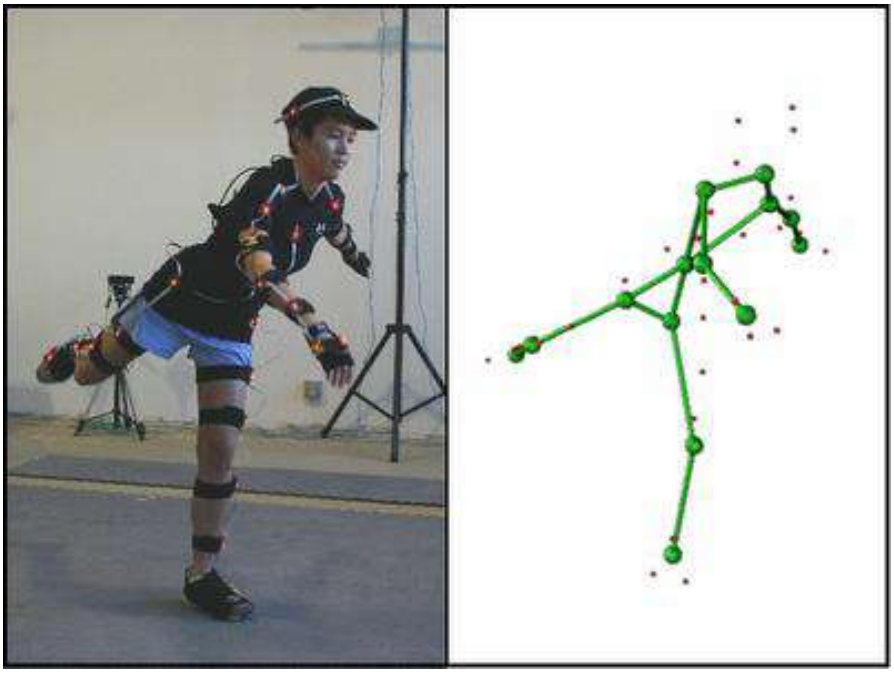


Рисунок 11.69 – Оптично активні датчики

Мінуси активних систем:

- відсутність можливості захоплення руху і міміки обличчя;
- додатковий контролер, що кріпиться до актора і підключений до маркерів-світлодіодів, сковує його рух;
- крихкість і відносно висока вартість маркерів-світлодіодів.

3. **Магнітні системи**, в яких маркерами є магніти, а ресиверами – камери, ткт система вираховує їхню позицію на основі спотворень магнітного потоку (рис. 11.70).

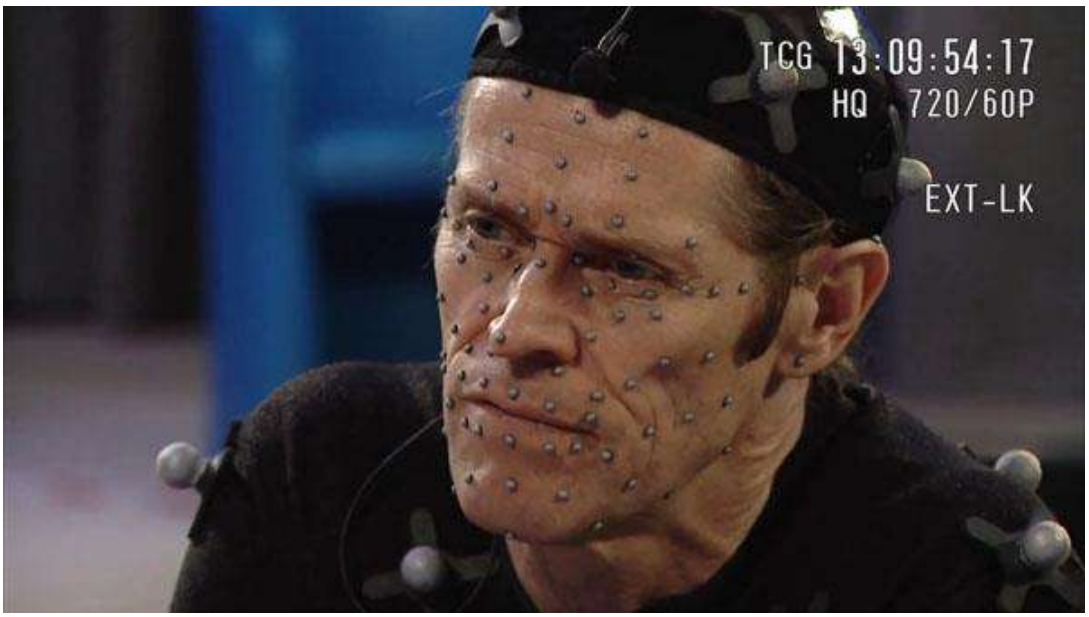


Рисунок 11.70 – Маркери розпізнавання міміки обличчя актора

Мінуси магнітних систем:

– магнітні системи схильні до магнітних та електричних перешкод від металічних предметів та оточення (електропроводки приміщення, оргтехніки, арматури в плитах будови);

– мінлива чутливість сенсорів залежно від їх положення у робочій зоні;

– в порівнянні з оптичними системами менша робоча зона;

– відсутність можливості захоплення рухів та міміки обличчя;

– додатковий контролер, прикріплений до актора і підключений до магнітних маркерів, або навіть зв'язка дротів, яка тягнеться від актора до комп'ютера;

– висока вартість магнітних маркерів.

4. **Механічні системи** безпосередньо слідкують за згинами суглобів, для цього на актора вдягається спеціальний механічний **Mo-Cap**-скелет, який повторює слідом за ним всі рухи. До комп'ютера при цьому передаються дані щодо кутів згинів всіх суглобів (рис. 11.71).

Рисунок 11.71 – Механічні системи **Мо-сар**

Мінуси механічних систем:

– **Мо-Сар**-скелет, з додатковим контролером, прикріплений до актора і підключений до сенсорів згинів, а в деяких випадках і дроти, які тягнуться від скелета, сильно обмежують рухи актора;

– відсутність можливості захвату:

– рухів та міміки обличчя;

– рухів близької взаємодії двох чи більше акторів (боротьба, танці з підтримками і т.д.);

– рухів на підлозі – перекиди, падіння і т.д.;

– ризик несправності механіки при необережному використанні.

5. **Гіроскопічні/інерціальні системи** для збирання інформації про рухи використовують мініатюрні гіроскопи та інерціальні сенсори, розташовані на тілі актора – так само, як і маркери чи магніти в інших **Мо-Сар** системах. Дані з гіроскопів і сенсорів передаються на комп'ютер, де відбувається їх обробка та запис (рис. 11.72). Система визначає не тільки положення сенсора, а також кут його нахилу.

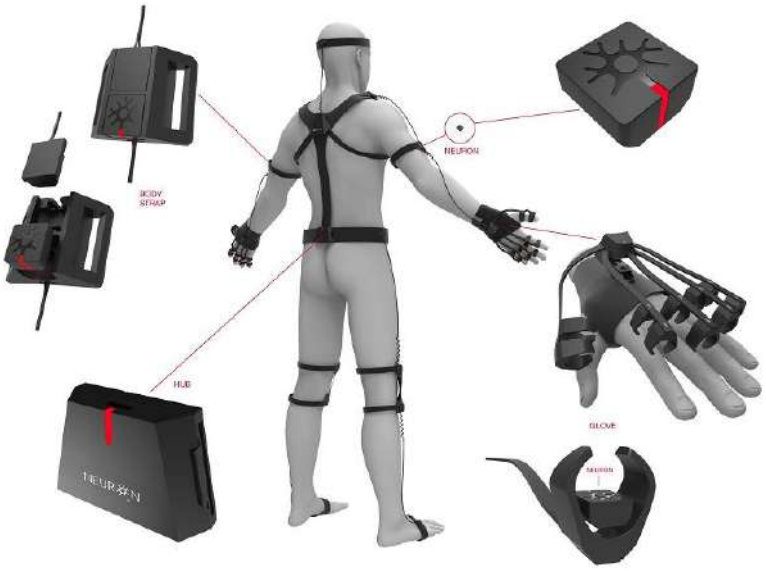


Рисунок 11.72 – Гіроскопічні/інерціальні системи **Mo-cap**

Мінуси гіроскопічних/інерціальних систем:

– відсутність можливості захоплення рухів та міміки обличчя;

– додатковій контролер, прикріплений до актора та підключений до магнітних маркерів, чи навіть зв'язка дротів, яка тягнеться від актора до комп'ютера;

– висока вартість гіроскопів та інерціальних сенсорів;

– для визначення положення актора у просторі потрібна додаткова міні-система (оптична чи магнітна).

В порівнянні з ***синім екраном*** **Motion Capture** має переваги:

– один актор може грати багато ролей;

– живе відео на тривимірному фоні може виглядати дещо чужорідно. Особливо це стосується 3D, виконуваного в реальному часі, на зразок комп'ютерних ігор;

– можливо редагування постфактум (зміна ракурсів, світла, незначне редагування рухів);

– більш широкі можливості *костюму* та *гриму*;

– можливість поєднання **Motion Capture** з ручною мультиплікацією;

– сцену можна показати з такого ракурсу, який навіть для павільйонних зйомок може мати складність;

– у сценах з великою кількістю комп'ютерних ефектів складно поєднати живих акторів з комп'ютерними персонажами.

Переваги «синього екрана»:

– більшість різновидів **Motion Capture** мають велику вартість;

– «синій екран» можна зробити досить великих розмірів та вести на його фоні зйомку масштабних сцен, в той час як розмір студії **Motion Capture** зазвичай обмежений;

– фотореалістичного персонажа зобразити на комп'ютері складніше, ніж фотореалістичне оточення. Тому, починаючи з деякої планки продуктивності, явно «комп'ютерний» персонаж буде чужерідно виглядати на фоні, який не відрізняється від реального.

Переваги ручної мультиплікації персонажів:

– більшість різновидів **Mo-Cap** мають високу вартість, комп'ютерна анімація дешевша;

– в **Mo-Cap** рухи персонажа обмежуються законами фізики;

– якщо анімована модель має інші пропорції, ніж актор, можливі проблеми. Наприклад, у «товстого» мультяшного персонажа, який анімований даними, ящо зняті навіть з дуже повної людини, руки можуть «входити» у тулуб;

– не завжди реалістичні рухи вдається добре пристосувати до комп'ютерної моделі (навіть ті, які мають звичайні людські пропорції). Взаємодія персонажа з крупними декораціями (наприклад, герой підходить до дверей та відкриває їх) в комп'ютерних іграх часто передається нереалістично. Крім того, часто в задум режисера входить «гіперболізація» рухів внаслідок меншої відповідності законам фізики. Наприклад, перш ніж *стрибнути*, людина робить особливий крок з присіданням. В комп'ютерних іграх цього кроку найчастіше немає, щоб не було затримки між натисканням кнопки і стрибком;

– труднощі з ручним редагуванням та «зшиванням» різних дублів порівняно з **Mo-Cap;**

– різноманітність – намальовані персонажі можуть виконувати набагато більше рухів, які люди повторити не зможуть.

Мультиплікатори порівнюють **Mo-Cap** з фотографією, а ручну мультиплікацію – з малюнком олівцем. Залежно від ситуації, перевага може бути віддана в рівній мірі тій чи іншій технології.

11.5. Візуалізація 3DS MAX. Mental Ray

Рендеринг/візуалізація – це процес растрування тривимірної векторної графіки у піксельну двовимірну графіку для формування растрового

зображення. Під растуванням розуміється процес перетворення векторних даних у двовимірні невекторні дані (рис. 11.73) подібно тому, як це зазвичай робиться при експорті векторної графіки у форматі растрових зображень.

Растрові зображення складаються з множини точок, які називаються *пікселями* (від **pixel - «picture element»** – елемент зображення).

Рисунок 11.73 – Порівняння векторної та растрової графіки

Векторні зображення складаються з сукупності простих геометричних елементів (*примітивів*, наприклад, кругів, прямокутників, відрізків і т.п.), де для кожного елемента визначені керуючі параметри. Наприклад, для прямокутника керуючим параметром є довжина, ширина, відстань по двох осях від верхньої лівої вершини до верхнього лівого кута зображення, колір, товщина лінії кордону, колір внутрішньої області.

Раструвати зображення – означає перетворити зображення, якіе складаються з множини фігур, заданих математичними алгоритмами, в зображення, яке складається з точок (пікселів).

11.5.1. Рендеринг за допомогою Mental Ray

Основне призначення **Mental Ray** складається у формуванні фотореалістичних зображень – процесі, який потребує складних розрахунків з використанням фізичних законів для імітації поведінки світла та його взаємодії з поверхнями. Також **Mental Ray** можна використовувати і для нефотореалістичної контурної візуалізації.

Mental Ray є незалежним рендерером (***рендерер*** – *програма візуалізації*), у якому реалізована можливість створення власної бібліотеки алгоритмів розрахунку зображення – шейдерів (***шейдер*** – *алгоритм, відповідальний за відповідну частину розрахунку зображення*). Ці шейдери програма використовує у комплексі (геометричну форму, матеріали, камери, світло) за допомогою власної мови опису сцен у форматі **Mental Images**.

У **Mental Ray** процес візуалізації виконується не порядково, як це відбувається під час рендерінгу **Scanline**, а за окремими мозаїчними елементами – бакетами (**Bucket**).

Під час рендерінгу 3DS Max сам створить файл опису сцени, перетворить матеріали, джерела світла і налаштування прорахунку в необхідні шейдери та запустить процес прорахунку в **Mental Ray.** Цей процес перетворення сцени також називається трансляцією. Трансляція є сполучною ланкою між 3DS MAX та Mental Ray.

11.5.2. Трасування променів

Ключовим алгоритмом, необхідним для прорахунку зображення, є ***процедура трасування променів***. Вона виконується однією з найперших на початку прорахунку. Трасування променів – це процес простеження шляхів від заданої точки в сцені і оцінки впливу на цю точку матеріалу інших точок з тривимірного оточення. Результатом є змінений колір точки, яку аналізують. Таким чином, можна прорахувати тіні, відображення, заломлення та пряму освітленість об’єктів (рис. 11.74).

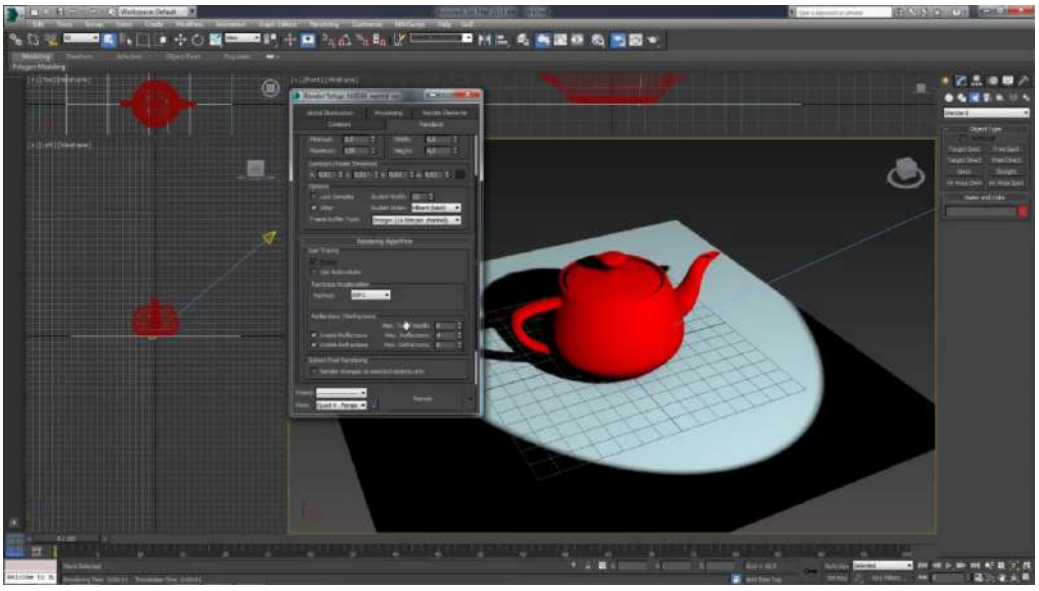


Рисунок 11.74 – Трасування променів

Існують два методи трасування променів: *пряме трасування променів* – це коли промені випускаються з джерел світла; і *зворотне трасування променів* – коли промені випускаються з камери (рис. 11.75).

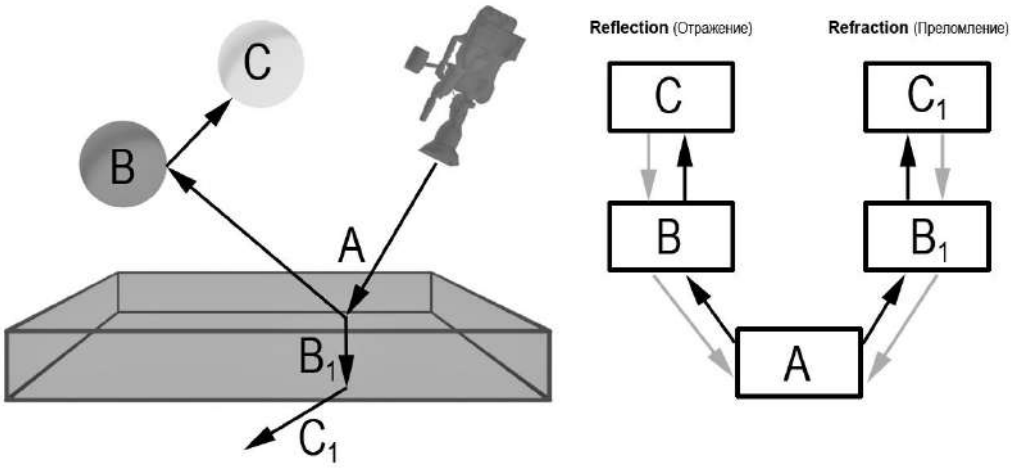


Рисунок 11.75 – Процес трасування променів

Коли промінь виходить з камери, він спочатку оцінюється в точці A, яка є вибірковою точкою. Отже, в точці A визначаються властивості поверхні, виходячи з параметрів налаштування матеріалу. Так, якщо поверхня віддзеркалює, переломлює світло або ж робить і те, і інше, то визначається, що потрібно відхилити промінь, а значить, направити у сцену вторинний промінь, щоб виявити об’єкти, які впливають додатково.

Якщо припустити, що в точці A відбувається віддзеркалення, то вторинний промінь слід направити в точку B, де процес визначення властивостей поверхні буде відбуватися спочатку. Якщо в точці B віддзеркалення не відбувається, то в результаті простих розрахунків повертається значення кольору для точки A. Але якщо в точці B віддзеркалення відбувається, то додатковий промінь направляється, наприклад, в точку C, і потім починається ще один розрахунок для точки C. Таким чином, з точки C повертається значення кольору віддзеркалення для точки B, а звідти – від точки A. Той самий алгоритм застосовується для розрахунку заломлених променів, які проходять крізь матеріал, на рисунку точки B_1 і C_1.

Головним завданням оптимізації є контроль часу процесу рендерінгу та якості зображення, що виключає такі артефакти, як ступінчастість та муар, методами семплінгу та фільтрації.

Приклад кількості трасування віддзеркалень поданий на рис. 11.76. В даному прикладі встановлено одне перевіддзеркалення променів, відповідно шари віддзеркалюються один в одному, але в їхньому віддзеркаленні вже немає взаємних перевіддзеркалень.

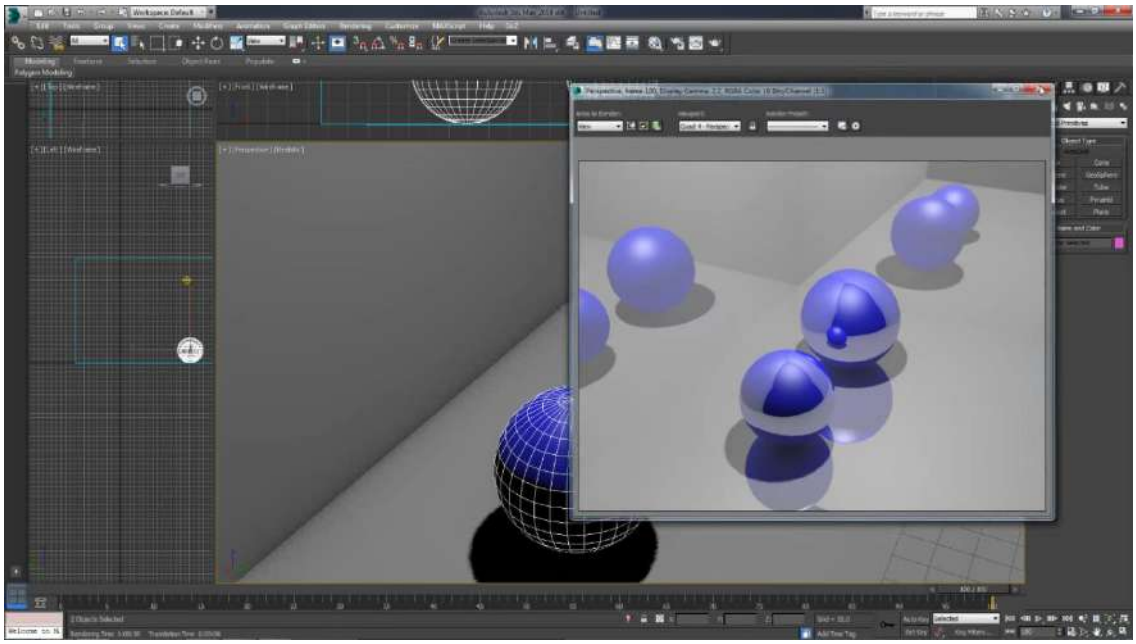


Рисунок 11.76 – Приклад кількості трасування віддзеркалень

Достатньо збільшити кількість віддзеркалень, і ми отримаємо наступну картину як на рис. 11.77.

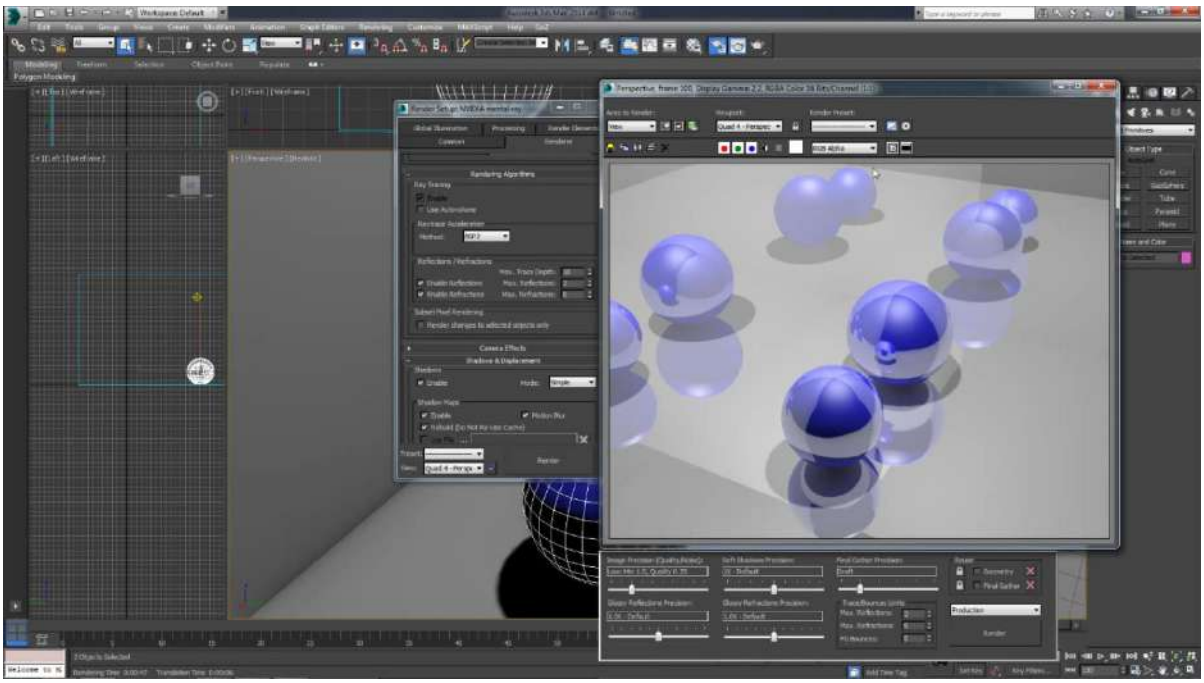


Рисунок 11.77 – Трасування віддзеркалень за умови, що **Max Raflections** дорівнює 2

Збільшення кількості віддзеркалень до 8 надасть наступний результат, найбільш наближений до реального сприйняття (рис. 11.78).

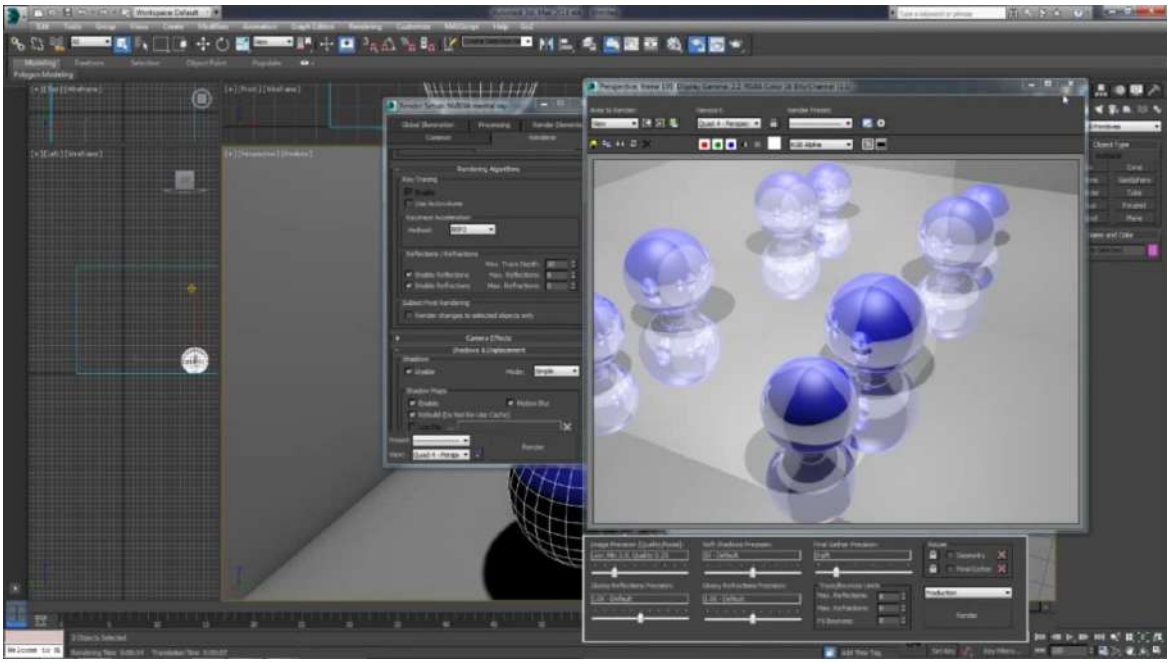


Рисунок 11.78 – Трасування віддзеркалень за умови, що Max Raflections дорівнює 8

Семплінг – перший етап для *визначення значення кольору* точки у сцені. В **Mental Ray** значення кольору обираються зі сцени за допомогою ряду променів в напрямку погляду при трасуванні променів. Після семплінгу застосовується фільтрація для усереднення обраних кольорів до кольорів окремих пікселів та визначення фінального значення кольору кожного пікселя, який направлений в буфер кадрів. Даний процес називається ***Згладжуванням (Antialiasing)***.

Згладжування слугує для отримання більш плавних переходів між границями об’єктів (рис. 11.69).

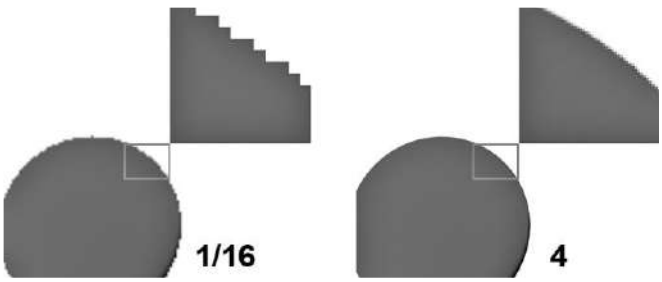


Рисунок 11.79 – Ступінчастість по краю сфери, 1/16 семплу на піксель (зліва); результат застосування згладжування, 4 семпли на піксель (справа)

В 3DS MAX, починаючи з версії 2009, **Mental Ray** є рендерером, установленим за замовчуванням. Для того щоб вибрати **Mental Ray** як програму рендерера, натисніть **F10** і в діалоговому вікні призначте систему рендерера **Mental Ray**.

11.5.3. Поняття та настройка гамми

Сутність проблеми темних візуалізованих зображень полягає в неоднакових значеннях гамми в отримуваному зображенні і монітора, на якому його переглядають.

Gamma (**Гамма**) – це ступінь нелінійності кольорового градієнта від найтемнішого до найсвітлішого значення. З математичної точки зору лінійною є **Gamma 1.0**, вона відповідає «ідеальному» монітору, який має лінійну залежність відображення від білого кольору до чорного. Але таких моніторів не існує, у реальних пристроях гамма нелінійна. Для сучасних моніторів значення гамми зазвичай складає від 1.6 до 2.2. Але для зручності прийнято вважати, що нелінійність кольорового градієнта на всіх моніторах дорівнює 2.2. Тому в більш нових версіях 3DS MAX за замовчуванням встановлено значення гамми 2.2 (рис. 11.80).

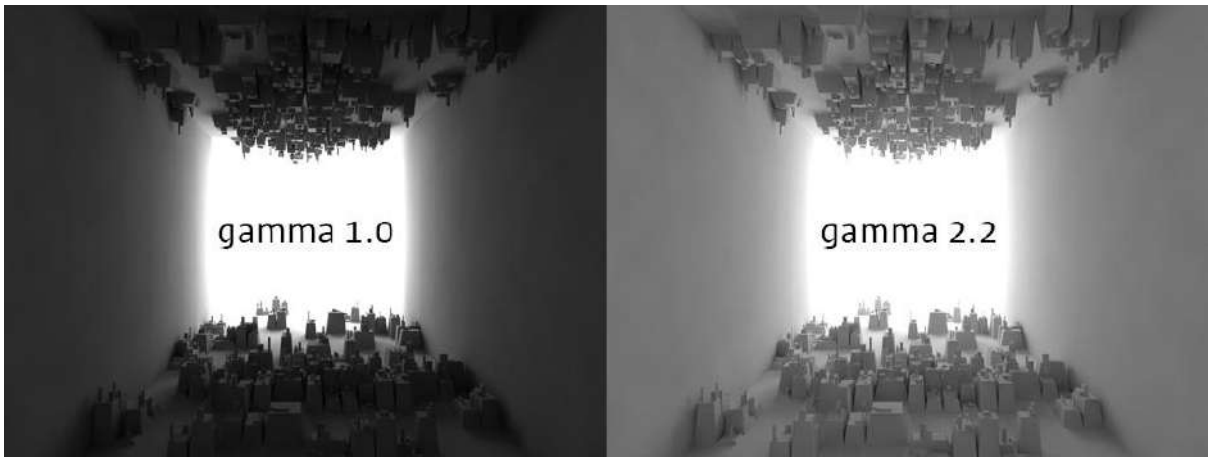


Рисунок 11.80 – Порівняння налаштувань відображення на моніторі G**amma 1.0** та **Gamma 2.2**

Для того щоб отримати зображення в гаммі, відповідній монітору (наприклад, **Gamma 2.2** (рис. 11.80 – 11.81)), варто змінити гамму вихідного зображення в налаштуваннях **Customize – Gamma and LUT**, або вже під час постобробки самого зображення у графічному редакторі Adobe Photoshop. У

другому варіанті час рендера буде значно менший, але якість опрацювання тіней також буде не максимальною.

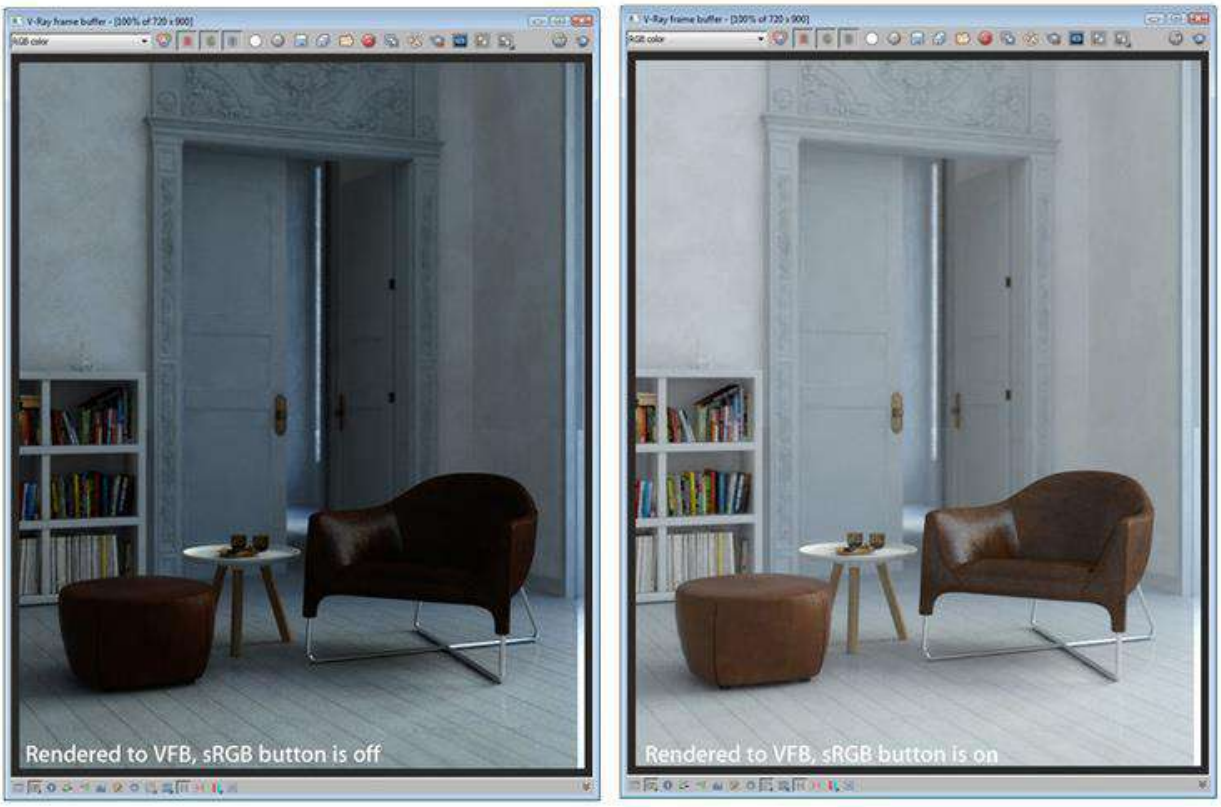


Рисунок 11.81 – Порівняння **Gamma 1.0** і **Gamma 2.2**

Якщо використовувати текстури у картах **Bump, Normal** і **Displacement**, то слід встановити гамму 1.0 у діалоговому вікні завантаження текстурних карт. Це необхідно для встановлення лінійної градації кольорів, для правильного розрахунку карт зміщень.

11.5.4. NVIDIA iray Renderer

IRAY рендерер (рис. 11.82) – це незалежна програма візуалізації, сумісна з **Mental Ray,** з деякими обмеженнями. **IRAY** може прораховувати зображення не тільки центральним процесором, але й з відеокартою на базі архітектури **CUDA**.

Рисунок 11.82 – Порівняння фотореалістичності рендера

Основними перевагами **IRAY** над **Mental Ray** є: відсутність проміжних чи додаткових етапів прорахунку (зображення виходить одразу максимально опрацьованим), друге – матеріали, які самостійно світяться, призначені об'єктам сцени (**Self-illumination**). Цими об'єктами, що світяться, можна освітити усю сцену, не використовуючи жодного фотометричного джерела світла. І третє, найбільш важлива перевага – глянсові напівматові віддзеркалення, які прораховуються дуже швидко та з високою якістю.

Сама якість візуалізації сцени задається часом рендера (**Time**) і кількістю ітерацій **Iterations (number of passes)** для кожного з пікселей зображення. Також є режим **Unlimited**, де світлові промені трасуються нескінченно довго і ви власноруч, на око, визначаєте, чи достатньо якісне ваше зображення відрендерено, та зупиняєте процес у будь-який час.

11.5.5. Додаткові опції рендерингу

Ці параметри визначають опції прискорення продуктивності.

Max Distance (Максимальна відстань) – дозволяє лімітувати віддзеркалення в межах певної дистанції, що прискорює рендеринг, а також дозволяє уникнути віддзеркалень предметів, які знаходяться далеко.

Fade to end color (Спад до кінцевого кольору) – при ввімкненні цієї опції градієнт віддзеркалень прагне до цього кольору. За вимкненої опції

градієнт віддзеркалень прагне до кольору віддзеркалень. Перша краща для інтер'єрних (закритих) сцен, а друга – для екстер'єрних (зовнішніх). Доступно, лише коли **Max Distance** ввімкнений.

Max Trace Depth (Максимальна глибина трасування) – коли глибина трасування відповідає зазначеному числу цього параметра, матеріал поводить себе, як наче були ввімкнені опції **Highlights+ FG Only**, тобто є тільки світіння та емулювання віддзеркалення .

Cutoff Threshold (Поріг зрізу) – пороговий рівень, за котрого вимикається трасування променів, відповідно і віддзеркалення також припиняється. Це відносне значення: наприклад, значення за замовчуванням 0.01 означає, що промені, які складають менше одного процента фінального пікселя, ігноруються. Значення 0.25 означає, що **Mental Ray** ігнорує промені, які складають менше чверті фінального пікселя.

11.6. Контрольні запитання і завдання

1. Як виконується анімація руху людини та інших персонажів у 3DS MAX?
2. Які складності виникають при анімації моделі персонажа?
3. Яким чином виконується контроль деформації полігональної сітки?
4. Які існують методики захвату рухів акторів?
5. Який алгоритм створення фізики взаємодії об'єктів у 3DS MAX?
6. Що таке спінінг?
7. За яким алгоритмом виконуються візуалізація та рендер сцени?

11.7. Завдання для самостійного розв'язання

Створити скелет чотирьох лапої тварини, обернути її у поверхні з полігональної сітки, та запустити тестову анімацію "біг".

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Хольцшлаг Молли. Использование HTML 4. Пер с англ.: учеб. пособие. – М.: Изд. дом «Вильямс», 2001. –1008 с.

2. Томсон Л., Веллинг Л. Разработка Web-приложений на PHP и MySQL: Пер. с англ. – К.: Изд-во «ДиаСофт», 2002. – 672 с.

3. Кингсли-Хью Э., Кингсли-Хью К. JavaScript в примерах: Пер. с англ. (Серия «Для программистов») – М.: ДМК Пресс. – 272 с.

4. Закас Н. JavaScript для профессиональных веб-разработчиков / [Пер. с англ. А. Лютича]. (Серия «Для профессионалов»). – СПб.: Питер, 2015. – 960 с.

5. Мейер Эрик, Уэйл Эстелл. CSS: полный справочник, 4-е изд.: Пер. с англ. Парал. тит. англ. – СПб.: ООО «Диалектика», 2019. – 1088 с.

6. Перепелица Ф. А. Разработка интерактивных сайтов с использованием jQuery. – СПб: Университет ИТМО, 2015. – 142 с.

7. Скворцов А. В. Триангуляция Делоне и ее применение. – Томск: Изд-во Том. ун-та, 2002. – 128 с.

8. Evans F., Skiena S., Varshney A. Optimizing triangle strips for fast rendering // Proc. IEEE Visualization. 1996. P. 319–326.

10. Autodesk 3DS MAX 9. Офіційний навчальний курс.: Пер. с англ. – К.: «МК-Пресс»,2007. – 528 с.: іл.

11. Верстак А. В. 3DS MAX 8. Секреты мастерства. – СПб.:Питер, 2006. – 672 с.: ил.

12. Лобанов Алексей. Mental Ray/Iray. Мастерство визуализации в Autodesk 3DS MAX. – ДМК Пресс, 2012.- 260 с.

13. Jamie Cardoso. Lighting La Salle + 3dsmax + Mental Ray. - 3dtotal.com, 2011. –86c.

14. Jennifer O'Connor. Mastering mental ray: Rendering Techniques for 3D and CAD Professionals. – Sybex, 2010 -370c.

15. Швембергер, С. И. 3DS MAX. Художественное моделирование и специальные эффекты / С. И. Швембергер. – СПб.: BHV, 2006. - 320 с.

16. Аббасов, И. Б. Двухмерное и трехмерное моделирование в 3DS MAX / И. Б. Аббасов. – М.: ДМК, 2012. - 176 с.

17. А. Потапкин 3D studio MAX / А. Потапкин, Д. Кучвальский. – М.: Эком, 2017. – 480 с.

18. Мортье, Шаммс 3DS MAX 14 для "чайников" (+ CD-ROM); М.: Вильямс – Москва, 2014. – 368 с.

19. Шнейдеров, Виталий Иллюстрированный самоучитель 3DS MAX; СПб: Питер – Москва, 2010. – 480 с.

Додаток А

ПСЕВДОКЛАСИ ТА ПСЕВДОЕЛЕМЕНТИ CSS

Псевдокласи – це особливі властивості, які дозволяють змінювати стиль елемента залежно від дій користувача, а також положення цього елемента (тега) в загальному потоці документа, що дозволяє розбавити дизайн сторінки певною динамікою і логікою. Класичним прикладом застосування псевдокласів є посилання, яке змінює свій колір з наведенням на нього курсора.

Список усіх псевдокласів:

– ***hover*** – стиль елемента, на який наведено курсор мишки;

– ***active*** – стиль для посилання, яке стає активним, але перехід по ньому ще не досконалий;

– ***visited*** – стиль для нещодавно відвіданих посилань;

– ***link*** – стиль для нечасто відвідуваного посилання;

– ***focus*** – стиль елемента, що знаходиться у фокусі;

– ***first-child*** – стиль першого дочірнього елемента;

– ***lang*** – вказати мову, що використовується у фрагменті документа.

Псевдоелементи – це особливий вид властивостей CSS, які дозволяють працювати не над самим елементом, а над його окремою частиною.

Перелік усіх псевдоелеметів:

– ***: first-letter*** – стиль першої літери текстового блоку;

– ***: first-line*** – стиль першого рядка текстового блоку;

– ***: after*** – додає вміст після елемента;

– ***: before*** – додає вміст до елемента;

– ***:: selection*** – стиль виділеного користувачем тексту.

Додаток Б

ПРАВИЛА АНІМАЦІЇ CSS

Правила	Приклад використання
1	2
Ключові кадри створюються за допомогою ключових слів from і to	***@keyframes move {*** ***from,*** ***to {*** ***top: 0;*** ***left: 0;*** ***}*** ***25%,*** ***75% {top: 100%;}*** ***50% {top: 50%;}*** ***}***
	h1 { ***font-size: 3.5em;*** ***color: darkmagenta;*** ***animation: shadow 2s infinite ease-in-out;*** ***}***
Часова функція для ключових кадрів	***@keyframes bounce {*** ***from {*** ***top: 100px;*** ***animation-timing-function: ease-out;*** ***}*** ***25% {*** ***top: 50px;*** ***animation-timing-function: ease-in;*** ***}*** ***50% {*** ***top: 100px;*** ***animation-timing-function: ease-out;*** ***}*** ***75% {*** ***top: 75px;*** ***animation-timing-function: ease-in;*** ***}*** ***to {*** ***top: 100px;*** ***}*** ***}}***

Продовження додатка Б

1	2
Властивість animation-name	***animation-name: none;*** ***animation-name: test-01;*** ***animation-name: -sliding;*** ***animation-name: moving-vertically;*** ***animation-name: test2;*** ***animation-name: test3, move4;*** ***animation-name: initial;*** ***animation-name: inherit;***
Тривалість анімації animation-duration	***animation-duration: .5s;*** ***animation-duration: 200ms;*** ***animation-duration: 2s, 10s;*** ***animation-duration: 15s, 30s, 200ms;***
Властивість animation-timing-function описує, як буде розвиватися анімація між кожною парою ключових кадрів. Під час затримки анимації часові функції не застосовуються	***animation-timing-function: ease-out;*** ***animation-timing-function: ease-in-out;*** ***animation-timing-function: linear;*** ***animation-timing-function: step-start;*** ***animation-timing-function: step-end;*** ***animation-timing-function: cubic-bezier (0.1, 0.7, 1.0, 0.1);*** ***animation-timing-function: steps (4, end);*** ***animation-timing-function: ease, step-start, cubic-bezier (0.1, 0.7, 1.0, 0.1);*** ***animation-timing-function: initial;*** ***animation-timing-function: inherit;***
Повтор анімації: властивість animation-iteration-count	***animation-iteration-count: infinite;*** ***animation-iteration-count: 3;*** ***animation-iteration-count: 2.5;*** ***animation-iteration-count: 2, 0, infinite;***
Напрямок анімації: властивість animation-direction	***animation-direction: normal;*** ***animation-direction: reverse;*** ***animation-direction: alternate;*** ***animation-direction: alternate-reverse;*** ***animation-direction: normal, reverse;*** ***animation-direction: alternate, reverse, normal;*** ***animation-direction: initial;*** ***animation-direction: inherit;***
Програвання анімації: властивість animation-play-state	***animation-play-state: running;*** ***animation-play-state: paused;*** ***animation-play-state: paused, running, running;*** ***animation-play-state: initial;*** ***animation-play-state: inherit;***

Продовження додатка Б

1	2
Затримка анімації: властивість animation-delay	***animation-delay: 5s;*** ***animation-delay: 3s, 10ms;***
Стан елемента до і після відтворення анімації: властивість animation-fill-mode	***animation-fill-mode: none;*** ***animation-fill-mode: forwards;*** ***animation-fill-mode: backwards;*** ***animation-fill-mode: both;*** ***animation-fill-mode: none, backwards;*** ***animation-fill-mode: both, forwards, none;***

Додаток В

ПЕРЕЛІК АНІМАЦІЇНИХ ТА ІНШИХ ВЛАСТИВОСТЕЙ CSS

-moz-outline-radius	flex-shrink	perspective
-moz-outline-radius-bottomleft	font	padding-right
-moz-outline-radius-bottomright	font-size	perspective-origin
-moz-outline-radius-topleft	font-size-adjust	right
-moz-outline-radius-topright	font-stretch	rotate
-webkit-line-clamp	font-variation-settings	row-gap
-webkit-text-fill-color	font-weight	scale
-webkit-text-stroke	gap	scroll-margin
-webkit-text-stroke-color	grid-column-gap	scroll-margin-block
all	grid-gap	scroll-margin-block-end
backdrop-filter	grid-row-gap	scroll-margin-block-start
background	grid-template-columns	scroll-margin-bottom
background-color	grid-template-rows	scroll-margin-inline
background-position	height	scroll-margin-inline-end
background-size	inset	scroll-margin-inline-start
border	inset-block	scroll-margin-left
border-bottom	inset-block-end	scroll-margin-right
border-bottom-color	inset-block-start	scroll-margin-top
border-bottom-left-radius	inset-inline	scroll-padding
border-bottom-right-radius	inset-inline-end	scroll-padding-block
border-bottom-width	inset-inline-start	scroll-padding-block-end
border-color	left	scroll-padding-block-start
border-end-end-radius	letter-spacing	scroll-padding-bottom
border-end-start-radius	line-clamp	scroll-padding-inline
border-image-outset	line-height	scroll-padding-inline-end
border-image-slice	margin	scroll-padding-inline-start
border-image-width	margin-bottom	scroll-padding-left
border-left	margin-left	scroll-padding-right
border-left-color	margin-right	scroll-padding-top
border-left-width	margin-top	scroll-snap-coordinate
border-radius	mask	scroll-snap-destination
border-right	mask-border	scrollbar-color
border-right-color	mask-position	shape-image-threshold
border-right-width	mask-size	shape-margin
border-start-end-radius	max-height	shape-outside
border-start-start-radius	max-lines	tab-size
border-top	max-width	text-decoration

Продовження додатка В

border-top-color	min-height	text-decoration-color
border-top-left-radius	min-width	text-decoration-thickness
border-top-right-radius	object-position	text-emphasis
border-top-width	offset	text-emphasis-color
border-width	offset-anchor	text-indent
bottom	offset-distance	text-shadow
box-shadow	offset-path	text-underline-offset
caret-color	offset-position	top
clip	offset-rotate	transform
clip-path	opacity	transform-origin
color	order	translate
column-count	outline	vertical-align
column-gap	outline-color	visibility
column-rule	outline-offset	width
column-rule-color	outline-width	word-spacing
column-rule-width	padding	z-index
column-width	padding-bottom	zoom
columns	padding-left	padding-right
filter	flex-basis	padding-top
flex	flex-grow	perspective
flex-basis	flex-shrink	perspective-origin
flex-grow	padding-top	

Додаток Г

КОДУВАННЯ КОЛЬОРУ У CSS

Назва кольору	HEX-код	Колір
1	2	3
AntiqueWhite	#FAEBD7	
Aqua	#00FFFF	
Aquamarine	#7FFFD4	
Black	#000000	
BlanchedAlmond	#FFEBCD	
Blue	#0000FF	
BlueViolet	#8A2BE2	
Brown	#A52A2A	
Chocolate	#D2691E	
Coral	#FF7F50	
CornflowerBlue	#6495ED	
Cornsilk	#FFF8DC	
Crimson	#DC143C	
Cyan	#00FFFF	
DarkBlue	#00008B	
DarkCyan	#008B8B	
DarkGoldenRod	#B8860B	
DarkGreen	#006400	
DarkKhaki	#BDB76B	
DarkMagenta	#8B008B	
DarkOliveGreen	#556B2F	
DarkRed	#8B0000	
DarkSalmon	#E9967A	
DarkSlateBlue	#483D8B	
DeepPink	#FF1493	
DeepSkyBlue	#00BFFF	
FireBrick	#B22222	
ForestGreen	#228B22	
Fuchsia	#FF00FF	
Gold	#FFD700	

Продовження додатка Г

1	2	3
GoldenRod	#DAA520	
Gray	#808080	
Green	#008000	
HotPink	#FF69B4	
IndianRed	#CD5C5C	
Indigo	#4B0082	
LawnGreen	#7CFC00	
LightBlue	#ADD8E6	
Lime	#00FF00	
LimeGreen	#32CD32	
Linen	#FAF0E6	
Magenta	#FF00FF	
Maroon	#800000	
MediumBlue	#0000CD	
MediumSeaGreen	#3CB371	
MediumSpringGreen	#00FA9A	
MediumVioletRed	#C71585	
MidnightBlue	#191970	
Olive	#808000	
Orange	#FFA500	
OrangeRed	#FF4500	
Purple	#800080	
Red	#FF0000	
SaddleBrown	#8B4513	
SlateBlue	#6A5ACD	
SpringGreen	#00FF7F	
Violet	#EE82EE	
Yellow	#FFFF00	

Додаток Д

ОПИС ГОЛОВНИХ ТЕГІВ

Тег	Опис
1	2
<a>	Тег для посилань
<p>	Параграф. Дозволяє вказати область тексту, яка належить одному абзацу
*** ***	Переклад рядка в будь-якому місці тексту, включаючи абзаци. Абзац не розірваний
*** ***	Невпорядкований список. Дозволяє позначити список, елементи якого, залежно від значення атрибута Type (застарілої), буде позначено як квадрат або коло
*** ***	Вказівковий тег
*** ***	Дозволяє позначити список, елементи якого будуть позначені серійним числом. У розмітці кожен пункт у списку відзначив ... </ li>
<h1> <h6>	Заголовний тег
<input>	Основний елемент у формі <input>, з типом поля типу, який визначається типом Тег <input>, є одним з різнобічних елементів форми і дозволяє створювати різні елементи інтерфейсу і забезпечити взаємодію з користувачем. Головним чином <input> призначений для створення текстових полів, різних кнопок, перемикачів і прапорців. Хоча елемент <input> не потрібно поміщати всередину контейнера <form>, що визначає форму, але якщо введені користувачем дані мають бути відправлені на сервер, де їх обробляє серверна програма, то вказувати <form> обов'язково. Те саме відбувається і в разі обробки даних за допомогою клієнтських додатків, наприклад, скриптів мовою JavaScript. Основний атрибут тега <input>, що визначає вид елемента – type. Він дозволяє задавати такі елементи форми: текстове поле (text), поле з паролем (password), перемикач (radio), прапорець (checkbox), приховане поле (hidden), кнопка (button), кнопка для відправленняформи (submit), кнопка для очищення форми (reset), поле для відправки файлу (file) і кнопка із зображенням (image). Для кожного елемента існує свій список атрибутів, які визначають його вид і характеристики. Крім того, в HTML5 додано ще більше десятка нових елементів. Приклад використання: <input type = "button | checkbox | file | hidden | image | password | radio | reset | submit | text">

Продовження додатка Д

1	2
<output>	Визначає область, в яку виводиться інформація, переважно за допомогою скриптів. Атрибути: for – визначає ідентифікатор одного і більше елементів для зв'язування з тегом <output>; form – задає ім'я форми, якій належить область для виведення; name – задає унікальне ім'я елемента
*** ***	Контейнер може містити вкладені елементи
<body>	Вміст цього тега відображається візуально на екрані
<meta>	Визначає метатеги, які використовуються для зберігання інформації, призначеної для браузерів і пошукових систем. Наприклад, механізми пошукових систем звертаються до метатегів для отримання опису сайту, ключових слів та інших даних: ***charset*** – задає кодування документа; ***content*** – встановлює значення атрибута, заданого за допомогою name або http-equiv; ***http-equiv*** – призначений для конвертації метатега в заголовок HTTP; ***name*** – ім'я метатега, також побічно встановлює його призначення
<Form>	Тег <form> встановлює форму на веб-сторінці. Форма призначена для обміну даними між користувачем і сервером. Область застосування форм не обмежена відправленням даних на сервер, за допомогою клієнтських скриптів можна отримати доступ до будь-якого елемента форми, змінювати його і застосовувати на власний розсуд. Документ може містити будь-яку кількість форм, але водночас на сервер може бути відправлена тільки одна форма. З цієї причини дані форм мають бути незалежні одне від одного. Наприклад: ***<form action = "handler.php">*** Тег має такі атрибути: ***accept-charset*** – встановлює кодування, в якому сервер може приймати та обробляти дані; ***action*** – адреса програми або документа, який обробляє дані форми; ***autocomplete*** – включає автозаповнення полів форми; ***enctype*** – спосіб кодування даних форми; ***method*** – метод протоколу HTTP; ***name*** – ім'я форми
<style>	Усередині даного тега задається стиль, оформлення сторінки
<table>	Задає таблицю

Додаток Е

ПЕРЕЛІК ФУНКЦІЇ JQWERY

Функція	Опис функції
.attr ()	повертає/змінює (залежно від кількості параметрів) значення атрибута в елементів на сторінці
.removeAttr ()	видаляє атрибут в елементів на сторінці
.addClass ()	додає клас елементів на сторінці
.removeClass ()	видаляє клас(и) в елементів на сторінці
.toggleClass ()	змінює наявність класу в елементів на протилежне (додає / видаляє)
.hasClass ()	перевіряє наявність класу з *Name* хоча б у одного з обраних елементів
.val ()	повертає/змінює (залежно від кількості параметрів) значення атрибута value в елементів на сторінці

РОБОТА З ПАРАМЕТРАМИ І СТИЛЯМИ

Стиль	Опис
.css ()	Повертає / змінює (залежно від кількості вхідних параметрів) CSS параметри елемента
.height () ***.innerHeight ()*** ***.outerHeight ()***	Повертає / змінює висоту елемента
.width () ***.innerWidth ()*** ***.outerWidth ()***	Повертає / змінює ширину елемента
.position () ***.offset ()***	Повертає / змінює позицію елемента
.offsetParent ()	Повертає найближчого предка з позиціонуванням, відмінним від static (позиціонування за замовчуванням)
.scrollTop () ***.scrollLeft ()***	Повертає / змінює величину скролінгу (прокрутку) елемента

ДОДАВАННЯ ВМІСТУ

Функція	Опис
1	2
.html ()	Повертає / змінює (залежно від кількості параметрів) html-вміст елементів на сторінці
.text ()	Повертає / змінює (залежно від кількості параметрів) текст, що знаходиться в елементах на сторінці
.append () ***.appendTo ()***	Додає заданий вміст у кінець елементів на сторінці

Продовження додатка Е

1	2
.prepend () ***.prependTo ()***	Додає заданий вміст у початок елементів на сторінці
.after () ***.insertAfter ()***	Додає заданий вміст після елементів на сторінці
.before () ***.insertBefore ()***	Додає заданий вміст перед елементами на сторінці
.wrap () ***.wrapAll ()***	Оточує елементи на сторінці заданими html-елементами
.wrapInner ()	Оточує вміст елементів на сторінці заданими html-елементами

ВИДАЛЕННЯ ВМІСТУ

Функція	Опис
.detach () ***.remove ()***	Видаляє елементи на сторінці
.empty ()	Видаляє вміст елементів на сторінці
.unwrap ()	Видаляє батьківські елементи, при цьому їх вміст залишається на місці

ЗАМІНА ЕЛЕМЕНТІВ

Функція	Опис
.replaceWith () ***.replaceAll ()***	Замінює одні елементи сторінки на інші (нові або вже існуючі)

КЛОНУВАННЯ ЕЛЕМЕНТІВ

Функція	Опис
.clone ()	Повертає копію обраних елементів сторінок

БАЗОВІ

Список функцій	Опис
1	2
.on ()	Універсальний метод для встановлення обробників подій на обрані елементи сторінки
.off ()	Видаляє обробники, встановлені з допомогою .on ()
.bind ()	Встановлює обробник події на обрані елементи сторінки. Оброблювач не спрацює на елементах, що з’явилися після його встановлення
.live ()	Встановлює обробник події на обрані елементи сторінки. Оброблювач спрацює також на елементах, що з’явилися після його встановлення

Продовження додатка Е

1	2
.delegate ()	Встановлює обробник події на обрані елементи сторінки. Елементи обираються за допомогою уточнюючого селектора. Оброблювач діятиме і на елементах, що з'явилися після його встановлення
.one ()	Встановлює обробник події на обрані елементи сторінки, який спрацює тільки по одному разу, на кожному з елементів
.unbind ()	Видаляє обробник подій в обраних елементах
.die ()	Видаляє обробник подій, який був встановлений за допомогою live ()
.undelegate ()	Видаляє обробник подій, який був встановлений за допомогою delegate ()
.trigger ()	Виконує вказану подію і запускає її обробник
.triggerHandler ()	Запускає обробник вказаної події, без її виконання
jQuery.proxy ()	За заданої функції створює іншу, всередині якої змінна this дорівнюватиме заданому значенню

ПОДІЇ МИШКИ

Подія	**Опис**
1	2
.click ()	Встановлює обробник «кліка» мишкою по елементу, або запускає цю подію
.dblclick ()	Встановлює обробник подвійного «кліка» мишкою по елементу або запускає цю подію
.hover ()	Встановлює обробник двох подій: появи / зникнення курсора над елементом.
.mousedown ()	Встановлює обробник натискання кнопки мишки або запускає цю подію
.mouseup ()	Встановлює обробник підняття кнопки мишки або запускає цю подію
.mouseenter ()	Встановлює обробник появи курсора в області елемента або запускає цю подію. Появу цієї події відпрацьовано краще, ніж стандартного mouseover
.mouseleave ()	Встановлює обробник виходу курсора з області елемента або запускає цю подію. Появу цієї події відпрацьовано краще, ніж стандартного mouseout
.mousemove ()	Встановлює обробник руху курсора в області елемента або запускає цю подію
.mouseout ()	Встановлює обробник виходу курсора з області елемента або запускає цю подію
.mouseover ()	Встановлює обробник появи курсору в області елемента або запускає цю подію

Продовження додатка Е

1	2
.toggle ()	По черзі виконує одну з двох або більше заданих функцій у відповідь на «клік» по елементу

ПОДІЇ КЛАВІАТУРИ

Подія	Опис
.keydown ()	Встановлює обробник переходу клавіші клавіатури у стан натиснутої або запускає цю подію
.keyup ()	Встановлює обробник повернення клавіші клавіатури у стан ненатиснутої або запускає цю подію
.keypress ()	Встановлює обробник введення символу з клавіатури або запускає цю подію

ПОДІЇ ФОРМИ

Подія	Опис
.focus ()	Встановлює обробник отримання фокусу або запускає цю подію.
.blur ()	Встановлює обробник втрати фокусу або запускає цю подію.
.focusin ()	Встановлює обробник отримання фокусу самим елементом, або одним з його дочірніх.
.focusout ()	Встановлює обробник втрати фокусу самим елементом, або одним з його дочірніх.
.select ()	Встановлює обробник виділення тексту або запускає цю подію.
.submit ()	Встановлює обробник відправки форми або запускає цю подію.
.change ()	Встановлює обробник зміни елемента форми або запускає цю подію.

ПОДІЇ ЗАВАНТАЖЕННЯ СТОРІНКИ

Подія	Опис
.ready ()	Встановлює обробник готовності дерева DOM
.load ()	Встановлює обробник завершення завантаження елемента
.unload ()	Встановлює обробник відходу зі сторінки (з переходом по посиланню, закриття браузера тощо)

ПОДІЇ БРАУЗЕРА

Подія	Опис
.error ()	Встановлює обробник помилки з завантаженням елементів (наприклад, відсутність необхідної картинки на сервері)
.resize ()	Встановлює обробник зміни розмірів вікна браузера або запускає цю подію
.scroll ()	Встановлює обробник «прокручування» елементів документа або запускає цю подію

Навчальне видання

ПОРОШИН Сергій Михайлович,
КАРТАШОВ Володимир Михайлович,
УСИК Вікторія Валеріївна,
ЦЕХМІСТРО Роман Іванович,
БЄЛІКОВ Ігор Сергійович

Технології створення складових мультимедійного контенту. Анімація та web-анімація

Навчальний посібник
для студентів усіх форм навчання за спеціальністю
123 «Комп'ютерна інженерія»

Відповідальний за випуск проф. Порошин С. М.
Роботу до видання рекомендував проф. Заполовський М. Й.
Редактор О. І. Шпільова

План 2021 р., поз.102.

Підписано до друку 31.01 .2022 р. Формат 60 × 84 1/16.
Папір офсетний. RISO-друк. Гарнітура Тайм. Обл.– вид. арк. – 19,6
Наклад 100 прим. Зам. № ___. Ціна договірна

Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК № 5478 від 21.08.2017 р.
61002, Харків, вул. Кирпичова, 2

Видавництво та друк ФО-П Єфименко С.А.
61166, Україна, м. Харків, вул. Коломенська,27
Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру видавців - виготовлювачів і розповсюджувачів видавничої продукції
ДК № 6869 від 08.08.2019 р.