KMS

Page

1. Принципы дискретизации и квантования для оцифровки мультимедийной информации.

Для оцифровки мультимедийной информации, такой как звук или изображение, используются два ключевых процесса: **дискретизация** и **квантование**. Эти процессы преобразуют непрерывные аналоговые сигналы в цифровую форму, пригодную для хранения, обработки и передачи в цифровых системах. Рассмотрим их основные принципы.

Дискретизация
Дискретизация — это процесс преобразования непрерывного сигнала в последовательность отсчётов, взятых в определённые моменты времени. Этот процесс фиксирует значения сигнала через равные интервалы времени.

- **Частота дискретизации** (или частота выборки): определяет, с какой скоростью берутся отсчёты. Согласно теореме Котельникова-Шеннона, частота дискретизации должна быть как минимум вдвое выше максимальной частоты спектра исходного сигнала (f >2f f >2f). Это необходимо для того, чтобы избежать эффекта наложения спектров (aliasing) и обеспечить возможность точного восстановления сигнала.
- Пример: Для аудио CD используется частота дискретизации 44,1 кГц, что позволяет корректно записывать звуки с частотой до 20 кГц (верхняя граница слышимого диапазона человека)

1. **Ограничение полосы частот**: перед дискретизацией применяется фильтр нижних частот для удаления компонент сигнала с частотами выше половины частоты дискретизации.
2. **Выборка значений**: устройство выборки-хранения фиксирует значения сигнала через равные временные интервалы.

Квантование
Квантование — это процесс округления значений отсчётов сигнала до

ближайшего уровня из заранее заданного набора фиксированных уровней. Оно преобразует амплитуду отсчётов в дискретные значения.

- **Глубина квантования**: определяется количеством битов, используемых для представления одного отсчёта. Например, глубина квантования 16 бит позволяет использовать 216=65 536216=65536 уровней квантования. Чем больше уровней, тем точнее представление амплитуды сигнала.
- **Шаг квантования**: интервал между соседними уровнями квантования. Меньший шаг обеспечивает более точное представление сигнала, но увеличивает объём данных[

Особенности:

1. **Шум квантования**: неизбежная ошибка, возникающая из-за округления значений. Он проявляется как слабый шум в сигнале.
2. Для уменьшения шума квантования могут использоваться дополнительные методы, такие как добавление небольшого случайного шума (dithering)

Связь между дискретизацией и квантованием
Дискретизация отвечает за временное представление сигнала (временные отсчёты), а квантование — за амплитудное представление (значения этих отсчётов). В результате их совместного применения аналоговый сигнал преобразуется в последовательность цифровых данных. Пример: В аудио CD сигнал записывается с частотой дискретизации 44,1 кГц и глубиной квантования 16 бит на канал. Это означает, что каждую секунду фиксируется 44 100 отсчётов, каждый из которых представлен одним из 65 536 возможных значений

Принципы дискретизации и квантования лежат в основе цифровой обработки мультимедийной информации. Правильный выбор параметров (частоты дискретизации и глубины квантования) позволяет достичь компромисса между качеством оцифровки и объёмом данных. Эти процессы обеспечивают возможность точного восстановления аналогового сигнала при воспроизведении и являются основой современных мультимедийных технологий

2. Общая структура форматов сжатия мультимедийной информации: JPEG для графики и MPEG для видео.

Сжатие мультимедийной информации в форматах **JPEG** (для графики) и **MPEG** (для видео) основано на устранении избыточности данных и использовании особенностей человеческого восприятия. Рассмотрим их общую структуру и основные принципы.

Формат JPEG
JPEG (Joint Photographic Experts Group) — это формат сжатия изображений, который применяется для фотографий и графики. Основной принцип JPEG заключается в сжатии с потерями, что позволяет значительно уменьшить размер файла, сохраняя приемлемое качество изображения.

Общая структура JPEG

1. **Разделение изображения на блоки:**
    - Изображение разбивается на блоки размером 8×8 пикселей.
    - Каждый блок обрабатывается независимо, что упрощает вычисления.
2. **Дискретное косинусное преобразование (DCT):**
    - Преобразует данные из пространственной области (пиксели) в частотную область.
    - Высокочастотные компоненты (детали, которые менее важны для восприятия) подвергаются более сильному сжатию или удаляются.
3. **Квантование:**
    - Коэффициенты после DCT округляются до ближайших значений из заранее определённой таблицы квантования.
    - Это снижает объем данных, но приводит к потере качества.
4. **Энтропийное кодирование:**
    - Используются методы сжатия без потерь, такие как кодирование Хаффмана или арифметическое кодирование, для дальнейшего уменьшения размера файла.
5. **Сохранение:**
    - Сжатые данные записываются в файл вместе с метаданными (например, информация о разрешении).

Особенности JPEG

- Поддерживает настройку степени сжатия: можно выбрать баланс между качеством изображения и размером файла.
- Недостаток: при высоком уровне сжатия появляются артефакты, такие как блочность.
- Применяется для фотографий и веб-графики благодаря высокой эффективности.

Формат MPEG
MPEG (Moving Picture Experts Group) — это семейство стандартов для сжатия видео и аудио. MPEG использует методы сжатия с потерями, оптимизируя хранение и передачу мультимедийных данных.

Общая структура MPEG

1. **Типы кадров:**
    - **I-кадры (Intra-coded frames):** Сжимаются как статические изображения (аналогично JPEG). Они являются ключевыми кадрами и обеспечивают случайный доступ к видео.
    - **P-кадры (Predicted frames):** Кодируются на основе предыдущих I- или P-кадров. Содержат только изменения относительно предыдущего кадра.
    - **B-кадры (Bidirectional frames):** Используют информацию как от предыдущих, так и от последующих кадров для максимального сжатия.
2. **Группы кадров (GOP — Group of Pictures):**
    - Кадры объединяются в группы, например, последовательность IBBPBBPBB.
    - GOP обеспечивает баланс между качеством, степенью сжатия и возможностью случайного доступа.
3. **Межкадровое сжатие:**
    - Устраняется временная избыточность между последовательными кадрами.
    - Анализируются изменения в макроблоках (16×16 пикселей), сохраняются только данные о движении или различиях.
4. **Внутрикадровое сжатие:**
    - Каждый кадр делится на макроблоки, которые обрабатываются аналогично JPEG (DCT → квантование → энтропийное кодирование).
5. **Синхронизация потоков:**
    - Видео- и аудиоданные синхронизируются для корректного воспроизведения.
    - Используются временные метки для согласования аудио- и видеопотоков.
6. **Метаданные:**
    - Файлы содержат информацию о структуре видео, разрешении, частоте кадров и других параметрах.

Особенности MPEG

- Высокая степень сжатия достигается за счёт комбинирования внутрикадрового и межкадрового методов.
- Поддерживает различные версии:
    - MPEG-1: Для VCD и базового видео.
    - MPEG-2: Для DVD и цифрового телевидения.
    - MPEG-4: Для потокового видео в интернете.
- Недостаток: сложность декодирования из-за зависимости B-кадров от соседних кадров.

Сравнение форматов

Характеристика	JPEG	MPEG
Тип данных	Графика	Видео
Метод сжатия	С потерями	С потерями
Основной алгоритм	DCT + квантование	Внутрикадровое + межкадровое сжатие
Единица обработки	Блок 8×8 пикселей	Кадры + макроблоки
Применение	Фотографии, веб-графика	Видео для VCD/DVD/ интернета
Настройка качества	Регулируемая степень компрессии	Зависит от структуры GOP

Оба формата используют особенности человеческого восприятия для эффективного хранения мультимедийных данных. JPEG идеально подходит для изображений, где допустима некоторая потеря качества, а MPEG обеспечивает высокую степень компрессии видео за счёт анализа временной избыточности между кадрами.

3. Правила создания и использования объектов «Кнопка» в среде Adobe Animate.

Создание и использование объектов «Кнопка» в Adobe Animate позволяет добавлять интерактивность в мультимедийные проекты. Кнопки представляют собой специальные символы, которые реагируют на действия пользователя, такие как наведение курсора, нажатие или отпускание. Рассмотрим основные правила их создания и использования.

Шаги для создания стандартной кнопки

1. **Создание символа кнопки:**
    - Выберите пункт меню **«Вставка» → «Создать символ»** или нажмите комбинацию клавиш `Ctrl+F8` (Windows) / `Command+F8` (Mac).
    - В диалоговом окне выберите тип символа **«Кнопка»**, задайте имя и нажмите «ОК».
2. **Редактирование состояний кнопки:**
    - После создания кнопки временная шкала переключается в режим редактирования, где отображаются четыре состояния:
        - **Up (Вверх):** состояние кнопки по умолчанию (когда она неактивна).
        - **Over (Наведение):** состояние при наведении курсора.
        - **Down (Нажатие):** состояние при нажатии кнопки.
        - **Hit (Активная область):** невидимая область, определяющая, где кнопка реагирует на действия пользователя.
    - Используйте инструменты рисования или импортируйте графику для каждого состояния.
3. **Добавление активной области (Hit):**
    - На кадре **Hit** создайте область взаимодействия с кнопкой. Это может быть прямоугольник или другая форма, соответствующая зоне клика.
4. **Применение анимации:**
    - Для создания анимации состояний можно использовать символы фрагментов роликов внутри состояний кнопки.

Назначение действий кнопке

Использование ActionScript

- Чтобы сделать кнопку интерактивной, необходимо добавить код на временную шкалу:1. Выберите кнопку на рабочей области.
  2. Откройте панель действий (`F9`) и добавьте код ActionScript.
  3. Пример простого кода:actionscript
  `myButton.addEventListener(MouseEvent.CLICK, handleClick);`

`function handleClick(event:MouseEvent):void { trace("Кнопка нажата!"); }`

- Для упрощения можно использовать готовые фрагменты кода из панели «Фрагменты кода».

Редактирование и тестирование кнопок

1. **Редактирование:**
    - Для изменения кнопки дважды щелкните по её экземпляру на рабочей области.
    - Используйте инспектор свойств для настройки параметров.
2. **Тестирование:**
    - Включите режим тестирования кнопок через меню **«Управление» → «Разрешить использование простых кнопок»**.
    - Для проверки работы кнопок используйте команду **«Управление» → «Тестировать клип»** (`Ctrl+Enter`).

Типы и особенности кнопок

- **Стандартные кнопки:** Используются для простых взаимодействий с пользователем.
- **Анимированные кнопки:** Для создания динамических эффектов можно использовать символы фрагментов роликов внутри состояний (например, анимацию при наведении).
- **Невидимые кнопки:** Состояния Up, Over и Down остаются пустыми, а активная область задаётся только в Hit-кадре.

4. Понятие о кватернионах, использование кватернионов для программирования поворотов 3D-объектов.

Кватернионы — это математический инструмент, который помогает описывать **вращения объектов в 3D-пространстве**. Они похожи на числа, но состоят из четырёх компонентов: w,x,y,zw,x,y,z. Эти компоненты можно представить как комбинацию одного обычного числа (ww) и трёх направлений (x,y,zx,y,z), которые задают ось вращения.

Почему кватернионы удобны для 3D?

1. **Нет блокировки гимбала (Gimbal Lock):** Это проблема, которая возникает при использовании углов Эйлера (например, углы поворота вокруг осей X, Y и Z). Кватернионы её избегают.

2. **Плавное вращение:** Кватернионы позволяют плавно переходить от одного поворота к другому (например, для анимации).
3. **Компактность:** Они занимают меньше памяти и быстрее вычисляются по сравнению с матрицами вращения.

Как работают кватернионы?
Кватернион для вращения состоит из двух частей:

- **Угол вращения** (θθ) — на сколько градусов нужно повернуть объект.
- **Ось вращения** (x,y,zx,y,z) — вокруг какой оси происходит поворот.

5. Представление базы знаний в виде семантической сети, объекты, субъекты и действия.

Семантическая сеть — это способ представления знаний, который используется для моделирования предметной области. Она представляет собой **ориентированный граф**, где:

- **Вершины** графа соответствуют объектам, понятиям, событиям или свойствам.
- **Дуги** (рёбра) графа обозначают отношения между этими вершинами.

Семантическая сеть позволяет структурировать информацию и связывать её элементы с помощью логических отношений, что делает её удобным инструментом для работы с базами знаний.

Основные элементы семантической сети

1. **Объекты (понятия):**
    - Это основные элементы предметной области. Они могут быть:
        - **Обобщёнными**: классы объектов (например, "животное", "транспорт").
        - **Индивидуальными**: конкретные экземпляры (например, "кот Василий", "автомобиль Tesla").
        - **Агрегатными**: составные объекты, состоящие из других элементов (например, "компьютер" как совокупность процессора, памяти и экрана).
2. **Субъекты:**
    - Субъектами в сети выступают агенты или участники действия. Например, в предложении "Иван читает книгу" субъектом является "Иван".

3. **Действия (события):**
    - Это процессы или изменения состояния объектов. Например, действие "читать" связывает субъекта "Иван" с объектом "книга".
    - Действия могут быть дополнены свойствами: временем выполнения, местом и т.д.
4. **Свойства:**
    - Характеристики объектов или событий. Например, у объекта "яблоко" может быть свойство "цвет: красный", а у действия "полететь" — свойство "время: утро".

6. Принцип членения предложений в тексте на триады для записей в базу знаний.

Принцип членения предложений в тексте на триады для записи в базу знаний
Членение предложений на триады — это метод структурирования информации, который используется для представления знаний в виде семантических сетей или RDF-троек. Основная идея заключается в том, чтобы каждое предложение разбить на три компонента: **субъект**, **предикат** и **объект**. Такой подход позволяет формализовать информацию и записывать её в базы знаний для дальнейшей обработки.

Что такое триада?
Триада — это минимальная единица представления знания, которая описывает отношение между двумя объектами или понятиями. Она состоит из трёх элементов:

1. **Субъект (Subject):** Кто или что является главным участником действия.
2. **Предикат (Predicate):** Отношение или действие, связывающее субъект с объектом.
3. **Объект (Object):** На что направлено действие или с чем связано отношение.

Пример:

Предложение: *"Анна читает книгу."*

Триада:

- Субъект: Анна
- Предикат: читает
- Объект: книга

Принципы членения предложений

1. **Актуальное членение предложения:**
    - Каждое предложение делится на **тему** (что уже известно) и **рему** (новая информация)
    - Тема обычно становится субъектом, а рема помогает определить предикат и объект.
2. **Семантический анализ:**
    - Выделяются основные смысловые элементы предложения: кто выполняет действие, какое действие выполняется и над чем оно совершается.
    - Например, в предложении *"Робот переместил коробку на склад"*:
        - Субъект: робот
        - Предикат: переместил
        - Объект: коробка
        - Дополнительная информация может быть записана как отдельные триады (например, "коробка находится на складе").
3. **Упрощение сложных предложений:**Предложение: *"Если станок закончил обработку, робот грузит кассету."*
    - Сложные предложения разбиваются на несколько простых триад.
    - Пример:

- Станок → закончил → обработку
- Робот → грузит → кассету

Триады:

1. **Использование стандартных отношений:**Предложение: *"Книга написана Толстым."*
    - Для унификации данных используются заранее определённые типы отношений (например, "является", "имеет", "содержит", "выполняет").
    - Пример:

- Книга → написана → Толстым

Применение триад в базах знаний

Триада:

1. **Семантические сети:**

- Узлы: Иван, яблоко, Мария
- Рёбра:

- Иван → купил → яблоко
- Мария → продала → яблоко

Триады используются для построения графов знаний, где узлы — это субъекты и объекты, а рёбра — предикаты
Пример сети для предложения *"Иван купил яблоко у Марии."*:

7. Алгоритм генерации семантической сети из сложных текстов информационной системы.

Генерация семантической сети из сложных текстов предполагает автоматическое преобразование текста в граф знаний, где узлы представляют объекты, понятия или события, а рёбра — отношения между ними. Это позволяет структурировать информацию для её дальнейшего использования в информационных системах, таких как базы знаний или интеллектуальные поисковые системы.

Основные этапы алгоритма

1. **Предварительная обработка текста**
   На этом этапе текст подготавливается для анализа:

- **Удаление ненужных элементов:** Исключение сокращений, оборотов, не несущих смысловой нагрузки (например, вводных слов).
- **Разбиение текста на предложения:** Используются знаки препинания (точки, запятые) для определения границ предложений.
- **Замена местоимений:** Местоимения заменяются на конкретные объекты для устранения неоднозначности (например, "он" заменяется на "Иван").

1. **Морфологический анализ**

- Определяются части речи каждого слова в предложении (существительное, глагол, прилагательное и т.д.).
- Используются словари частей речи и синонимов для унификации терминов.

1. **Синтаксический анализ**

- Выявляются семантически значимые элементы предложения: подлежащее, сказуемое, дополнение.
- Сложные предложения разбиваются на простые блоки с помощью синтаксического анализа (например, "Иван купил книгу и подарил её Марии" делится на два блока: "Иван купил книгу" и "Иван подарил книгу Марии").

1. **Актуальное членение предложений**

- Каждое предложение делится на триаду: **субъект**, **предикат**, **объект**.
    - Пример: "Иван читает книгу."
        - Субъект: Иван
        - Предикат: читает
        - Объект: книга

1. **Формирование семантических отношений**
   На основе синтаксического анализа создаются связи между элементами:

- Типы связей:
    - "является" (классификация): Например, "Иван является человеком".
    - "имеет" (атрибуты): Например, "Книга имеет автора".
    - "выполняет" (действия): Например, "Иван читает книгу".
- Связи добавляются в сеть в виде дуг графа

1. **Создание структуры графа**

- Узлы графа соответствуют объектам или понятиям.
- Рёбра графа представляют отношения между узлами.
- Пример сети для текста "*Иван читает книгу. Книга написана Толстым.*":
    - Узлы: Иван, книга, Толстой
    - Рёбра:
        - Иван → читает → книга
        - Книга → написана → Толстым

8. - 19. База знаний на JavaScript

Plain Text

```
var endings = [

["ет","(ет|ут|ют)"], ["ут","(ет|ут|ют)"], ["ют","(ет|ут|ют)"],
//1 спряжение
["ит","(ит|ат|ят)"], ["ат","(ит|ат|ят)"], ["ят","(ит|ат|ят)"],
//2 спряжение

["ется","(ет|ут|ют)ся"],["утся","(ет|ут|ют)ся"],["ются","(ет|ут|
ют)ся"],    //1 спряжение, возвратные
["ится","(ит|ат|ят)ся"], ["атся","(ит|ат|ят)ся"], ["ятся","(ит|ат|
ят)ся"],  //2 спряжение, возвратные
```

```
["ен","ен"], ["ена","ена"], ["ено","ено"], ["ены","ены"],
//краткие прилагательные
["ан","ан"], ["ана","ана"], ["ано","ано"], ["аны","аны"],
//краткие прилагательные
["жен","жен"], ["жна","жна"], ["жно","жно"], ["жны","жны"],
//краткие прилагательные

["такое","- это"]                                                    //
дополнение в массив псевдоокончаний
];
```

2.МАССИВ blacklist[ ] – «черный» список подлежащих в вопросе, имеющих совпадающие окончания с псевдоокончаниями сказуемых

```
var blacklist =
["замена", "замены", "атрибут", "маршрут", "член", "нет"];
```

ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ

1.Функция getEnding()для поиска сказуемого в вопросе по псевдоокончаниям

Для анализа сказуемого на совпадения его окончания с соответствующим ему псевдоокончанием в массиве псевдоокончаний endings[] определим вспомогательную функцию getEnding(word), в которой вначале сразу же проверяется, не находится ли рассматриваемое слово word в списке исключений массива blacklist[] и поэтому его нужно исключить из дальнейшего рассмотрения:

```
function getEnding(word)
{
//проверка слова на совпадение по черному списку в массиве
// blacklist

if (blacklist.indexOf(word)!==-1) return -1;
```

Затем в цикле для всех записей в первом столбце массива endings[]производится проверка, не имеет ли это слово одно из псевдоокончаний, характерных для сказуемого:

```
//перебор псевдоокончаний в массиве endings
for (var j = 0; j < endings.length; j++)
{

//проверка, оканчивается ли слово word на j-ое псевдоокончание
if(word.substring(word.length-endings[j][0].length)==endings[j][0])
```

```
//возврат номера найденного псевдоокончания для сказуемого
return j;
}

//если совпадений нет, то возврат -1
return -1;
}
```

2.Функция small()для преобразования первой буквы в тексте ответа в прописную

```
function small(str)
{
  return str.substring(0, 1).toLowerCase() + str.substring(1);
}
```

3.Функция big()для преобразования первой буквы в тексте ответа в заглавную

```
function big(str)
{
 return str.substring(0, 1).toUpperCase() + str.substring(1);
}

function getAnswer(question){
```

Параметром question для этой функции будет являться текст вопроса, который вводится в соответствующее окно при выполнении диалога с информационной системой.

1.В начале функции getAnswer() определим переменные для фиксации факта получения положительного результата result при посылке вопроса к БЗ и записи ответа на вопрос answer, который должен быть получен в результате выполнения функции:

```
//флаг, найден ли ответ на вопрос
var result = false;

//формируемый функцией ответ на вопрос – вначале пустой
var answer="";
```

2.Затем текст вопроса в параметре question готовится к обработке путем уменьшения первой буквы до прописной и отделения знаков препинания от слов вставкой между ними пробелов:

```
//преобразование текста в параметре question вспомогательной     //
функцией small()чтобы сделать первую букву в тексте
//вопроса прописной
```

```
var txt = small(question);

//знаки препинания в предложении вопроса
var separators = "'\",.!?()[]\\/";

//добавление пробелов перед знаками препинания

for (var i = 0; i < separators.length; i++)
txt = txt.replace(separators[i], " " + separators[i]);
```

3.После этого текст вопроса методом split() разбивается на отдельные слова, ориентируясь на все пробелы в тексте вопроса для его дальнейшего анализа, которые при этом сохраняются в массиве words, автоматически формируемым из текста этим методом:

```
//массив слов в вопросе, отделенных пробелами
var words = txt.split(' ');
```

Поскольку вопрос состоит из вопросного слова (вопросных слов), сказуемого и подлежащего, то сказуемое оказывается в центре вопроса, а если сказуемое в вопросе будет найдено, то искомое подлежащее будет следовать за ним, при этом вопросное слово игнорируется.

4.Теперь будем решать задачу поиска сказуемого в вопросе, для чего выполним цикл, перебирающем все слова в предложении вопроса, записанные в массив words. При этом текущее слово в массиве будет считаться сказуемым, если обладает характерным для сказуемых псевдоокончанием.

```
//перебор слов в массиве слов из вопроса

for (var i = 0; i < words.length; i++)
{
//поиск номера псевдоокончания с использованием //вспомогательной функции getEnding() с записью //найденного номера в переменную ending

var ending = getEnding(words[i]);
```

Если псевдоокончание будет найдено, а это эквивалентно возвращаемому значению функции getEnding() в виде номера в массиве, отличного от -1, то это и есть сказуемое в вопросе, а подлежащее в вопросе будет следовать сразу после него:

```
if (ending >= 0)
{
//замена псевдоокончания на набор возможных окончаний,
//хранящихся во втором столбце массива
```

```
words[i] =
words[i].substring(0,words[i].length-endings[ending][0].length)
 + endings[ending][1];
}
```

Таким образом, найденное в массиве words[i] сказуемое в вопросе заменяется выражением words[i], но уже с набором соответствующих ему псевдоокончаний из второго столбца массива endings[], что в конечном итоге превращает найденное в вопросе сказуемое в соответствующий ему шаблон – регулярное выражение для этого сказуемого.
Так например, любые из возможных окончаний сказуемых в вопросе для глаголов 1-го спряжения -ет, -ут, -ют будут заменены на общий шаблон с перечислением возможных вариантов строки в виде «(ет|ут|ют)».

Наконец, для поиска сказуемого в записях Базы знаний, соответствующего шаблону сказуемого в вопросе, необходимо сохранить полученный шаблон в переменную из класса RegExp().

```
//создание регулярного выражения по сказуемому из вопроса
var predicate = new RegExp(words[i]);
```

В случае, если в роли сказуемых выступают не глаголы, а краткие прилагательные, имеющие соответственно окончания типа –ен, -жен, -жно и т. п., которые в шаблоне в соответствии с таблицей псевдоокончаний сказуемых будут просто совпадать со словами с такими окончаниями, то в качестве сохраняемого значения для переменной predicate для обрабатываемого регулярного выражения необходимо захватить следующее слово за найденным в вопросе слова с окончанием, характерным для кратких прилагательных:

```
if (endings[ending][0] == endings[ending][1])
{
predicate = new RegExp(words[i] + " " + words[i + 1]);}
i++;
}
```

20. Назначение и общая структура функции dialog_window() для создания диалогового окна

Plain Text

```
function dialog_window(){

Добавляем на страницу диалоговое окно как новый раздел div с
идентификатором dialog, задаем ему CSS-стиль dialog, располагающий
этот раздел справа от основного раздела страницы, и затем надвигаем
его левое поле на страницу на 25px:

document.body.innerHTML+=
"<div id='dialog' class='dialog'>"

Здесь используется внешний CSS-стиль dialog:
.dialog{
 position:fixed;
 top:10%;
 left:100%;
 width:800px;
 height:800px;
 margin-left:-30px;
 margin-top:-25px;
 padding:10px;
 border:#000 2px solid;
 border-radius:10px;
 background:#fff;
 color:#f00;
 z-index:1500;
 font-size:12pt;
}


Не закрывая этот раздел, добавляем в него новый раздел со строкой
«Нажми, чтобы  спросить!», задаем ему CSS-стиль label, который уже
будет вложен в стиль  dialog родительского окна и команду на
выполнение функции openDialog() выдвижения диалогового окна щелчком
мыши:

+"<div class='label' onclick='openDialog()'>Нажми, чтобы спросить!
</div>"

Здесь используется внешний CSS-стиль .dialog. label:
.dialog .label{
 transform: rotate(270deg);
 width:430px;
 height:20px;
 text-align:center;
 color:#f00;
```

```
overflow:hidden;
 display:inline-block;
 margin-left:-26%;
 margin-top:50%;
 cursor:pointer;
}
```

23. Структура функции openDialog() плавного выдвижения-закрытия диалогового окна поверх Web-страницы.

Plain Text

```
function openDialog(){
if(dialogOn){   //анимация закрытия диалогового окна
$("#dialog").animate({"margin-left":"-30px"},1000,function(){});
dialogOn=false;
}
else{    //анимация открытия диалогового окна
$("#dialog").animate({"margin-left":"-1250px"},1000,function() {});
dialogOn=true;
clearInterval(timer);
}
}
```

24. Структура и стиль оформления блока div для выдачи ответа на вопрос в окно истории ответов.

Plain Text

```
.dialog .header{
 text-align:center;
 margin-top:-430px;
 font-size:20px;
}
```

Далее добавляем в раздел диалогового окна еще один новый раздел, задаем ему ему CSS-стиль history и задаем ему идентификатор history:

```
+"<div class='history' id='history'></div>"
```

Здесь используется внешний CSS-стиль .dialog .history:

```
.dialog .history{
 height: 730px;
 overflow-x:hidden;
 overflow-y:scroll;
}
```

Наконец, добавляем в диалоговое окно последний новый раздел с CSS-стилем question и вложенным в него полем формы input с идентификатором Qdialog и с текстом-приглашением 'Введите вопрос',исчезающим при вводе вопроса, а также в этот раздел в строке ниже включим кнопку с названием «Спросить» с командой для запуска функции ask(), которая обрабатывает заданный вопрос и выводит в раздел history полученный в результате обработки ответ:

```
+"<div class='question'><input id='Qdialog' placeholder='Введите
вопрос'/> <br><button onclick='ask(\"Qdialog\")'>Спросить</button>
</div>"
```

Здесь используется следующие CSS-стили:

```
.dialog .history .question{
 margin-left:40px;
 background:#99f;
margin:5px;
 width:100%;
 border-radius: 5px;
 padding:5px;
}
.dialog button{
 margin:2px 20px;
 width:95%;
 border:1px #666 solid;
 border-radius: 5px;
 font-size:20px;
}
input{
 font-size:20px;
 width:100%;
 border-radius:10px;
 padding:2px 10px;
}
```

25. Структура и стиль оформления блока div для поля формы input и кнопкой «Спросить» для запуска функции получения ответа из Базы знаний и выдачи его в блок ответа диалогового окна.

Plain Text

```
+"<div class='question'><input id='Qdialog' placeholder='Введите
вопрос'/> <br><button onclick='ask(\"Qdialog\")'>Спросить</button>
</div>"
.dialog .history .question{
 margin-left:40px;
 background:#99f;
margin:5px;
 width:100%;
 border-radius: 5px;
 padding:5px;
}
.dialog button{
 margin:2px 20px;
 width:95%;
 border:1px #666 solid;
 border-radius: 5px;
 font-size:20px;
}
input{
 font-size:20px;
 width:100%;
 border-radius:10px;
 padding:2px 10px;
}
```

Практика 1: Создание графического изображения с озвучиванием областей рисунка в Adobe Animate

1. **Создание графического изображения:**
    - Откройте Adobe Animate и создайте новый HTML5 Canvas проект.
    - Используйте инструменты рисования (например, «Кисть» или «Форма») для создания различных областей рисунка.

- Каждую область преобразуйте в символ, выбрав ее и нажав `F8`. Назовите каждый символ соответствующим образом (например, «Область1», «Область2» и т.д.).

2. **Добавление интерактивности:**
   - На сцене разместите экземпляры созданных символов.
   - Для каждого символа назначьте имя экземпляра в панели Свойств (например, `area1`, `area2`).
3. **Озвучивание областей:**
   - Импортируйте звуковые файлы, которые будут воспроизводиться при наведении курсора.
   - import flash.events.MouseEvent;

     area1.addEventListener(MouseEvent.MOUSE_OVER, playSound1);
     area1.addEventListener(MouseEvent.MOUSE_OUT, stopSound1);

     function playSound1(event:MouseEvent):void {
       var sound1:Sound = new Sound();
       sound1.load(new URLRequest("sound1.mp3"));
       sound1.play();
     }

     function stopSound1(event:MouseEvent):void {
       // Код для остановки звука, если необходимо
     }
   - Повторите этот процесс для каждой области, изменяя имена функций и звуковые файлы соответственно.
4. **Тестирование:**
   - Нажмите `Ctrl + Enter` для предварительного просмотра анимации.
   - Наведите курсор на каждую область и убедитесь, что звук воспроизводится корректно.

Практика 2: Создание анимационного ролика с кнопками запуска, остановки и перехода на начало в Adobe Animate

1. **Создание анимации:**
   - Создайте новую анимацию на временной шкале, используя классическую анимацию или анимацию с применением

инструментов «Кость» для риггинга персонажей.

2. **Создание кнопок:**
    - Создайте три кнопки: «Play», «Stop» и «Restart».
    - Используйте инструменты рисования для создания форм кнопок и преобразуйте их в символы типа «Кнопка».
    - Назначьте каждому экземпляру кнопки имя экземпляра (например, `playBtn`, `stopBtn`, `restartBtn`).
3. **Добавление интерактивности:**
    - ```
      // Остановка анимации на первом кадре
      stop();

      // Добавление событий для кнопок
      playBtn.addEventListener(MouseEvent.CLICK, playAnimation);
      stopBtn.addEventListener(MouseEvent.CLICK, stopAnimation);
      restartBtn.addEventListener(MouseEvent.CLICK, restartAnimation);

      function playAnimation(event:MouseEvent):void {
          play();
      }

      function stopAnimation(event:MouseEvent):void {
          stop();
      }

      function restartAnimation(event:MouseEvent):void {
          gotoAndPlay(1);
      }
      ```
4. **Тестирование:**
    - Нажмите `Ctrl + Enter` для проверки работы кнопок.
    - Убедитесь, что кнопки правильно запускают, останавливают и перезапускают анимацию.

Практика 3: Создание модели «танк-башня-ствол» в 3ds MAX с точками привязки и анимация движений

1. **Создание базовых примитивов:**
    - Запустите 3ds MAX и создайте примитивы для танка:
        - **Корпус:** Куб или цилиндр.
      ```

- **Башня:** Цилиндр меньшего размера, расположенный на корпусе.
- **Ствол:** Цилиндр, присоединенный к башне.

2. **Настройка точек привязки:**
    - Перейдите в режим иерархии (`Hierarchy`) и выберите «Сгруппировать».
    - Установите точки привязки для башни и ствола относительно корпуса, чтобы обеспечить правильное движение.
3. **Создание системы риггинга:**
    - Используйте систему костей (`Bones`) для создания скелета танка.
    - Свяжите примитивы с соответствующими костями для обеспечения плавного движения.
4. **Анимация движений:**
    - Перейдите в режим анимации и задайте ключевые кадры для вращения башни и поворота ствола.
    - Используйте инструменты анимации, такие как «Dope Sheet» и «Curve Editor», для настройки плавности движения.
5. **Тестирование:**
    - Воспроизведите анимацию, чтобы убедиться, что все части танка движутся синхронно и корректно.

Практика 4: Создание движения объекта по заданной траектории в 3ds MAX

1. **Создание объекта:**
    - Создайте объект, который будет двигаться (например, шар или куб).
2. **Определение траектории:**
    - Используйте инструмент «Path Constraint» для задания пути движения.
    - Нарисуйте кривую, представляющую желаемую траекторию.
3. **Применение ограничения:**
    - Примените «Path Constraint» к объекту, связывая его с созданной кривой.
    - Настройте параметры ограничения, такие как скорость движения и ориентация объекта по пути.
4. **Настройка анимации:**

- Установите ключевые кадры на временной шкале для начала и окончания движения.
- Используйте инструменты интерполяции для плавного следования объекта за траекторией.

5. **Тестирование:**
    - Воспроизведите анимацию, чтобы убедиться, что объект движется по заданной траектории корректно.

Практика 5: Редактирование объектов с использованием модификаторов и стека в 3ds MAX

1. **Создание объекта:**
    - Создайте примитивный объект, например, цилиндр.
2. **Применение модификаторов:**
    - Перейдите в модификаторный стек (`Modifier Stack`) и добавьте модификатор, например, «Twist».
    - Настройте параметры модификатора для достижения желаемого эффекта.
3. **Комбинирование модификаторов:**
    - Добавьте дополнительные модификаторы, такие как «Bend» или «FFD (Free-Form Deformation)», для дальнейшего редактирования формы объекта.
    - Порядок применения модификаторов влияет на конечный результат, поэтому экспериментируйте с последовательностью.
4. **Настройка параметров:**
    - Регулируйте параметры каждого модификатора для точной настройки формы объекта.
    - Используйте превью в режиме реального времени для оценки изменений.
5. **Закрепление изменений:**
    - После завершения редактирования примените модификаторы, чтобы закрепить изменения в объекте.

Практика 6: Использование освещения, материалов и текстур в 3ds MAX

1. **Создание сцены:**
    - Создайте несколько объектов в сцене, например, сферу и куб.
2. **Добавление освещения:**
    - Перейдите в меню освещения и добавьте источник света, например, «Omni Light».
    - Настройте параметры освещения, такие как интенсивность и цвет.
3. **Применение материалов:**
    - Откройте редактор материалов и создайте новый материал.
    - Настройте свойства материала, включая диффузный цвет, отражение и прозрачность.
4. **Добавление текстур:**
    - В разделе «Diffuse» добавьте текстурное изображение, выбрав нужную текстуру из файловой системы.
    - Настройте масштаб и расположение текстуры на объекте.
5. **Назначение материалов объектам:**
    - Примените созданный материал к выбранным объектам, перетащив его на них в сцене.
6. **Настройка камер:**
    - Добавьте камеру в сцену и настройте ее положение для оптимальной визуализации освещения и текстур.
7. **Рендеринг:**
    - Выполните рендеринг сцены, чтобы увидеть финальный результат с примененными освещением, материалами и текстурами.

Практика 7: Создание Web-страницы с адаптивным дизайном и эффектом фиксации заголовка

1. **Создание HTML-структуры:**
    - Создайте HTML-файл с базовой структурой, включая `<header>`, `<main>` и `<footer>`.
2. **Добавление стилей CSS:**
    - @media (max-width: 768px) {
      /* Стили для мобильных устройств */
      }

- Настройте гибкие сетки и использование относительных единиц измерения (`%, em, rem`) для элементов страницы.

3. **Фиксация заголовка:**
   - header {
     posit ion: fixed;
     top: 0;
     width: 100%;
     background-color: #fff;
     z-index: 1000;
     }
   - Добавьте отступ верхней части `<main>`, чтобы контент не скрывался за фиксированным заголовком.
4. **Добавление контента:**
   - Заполните страницу контентом, используя семантические теги HTML5.
5. **Тестирование адаптивности:**
   - Проверяйте отображение страницы на различных устройствах и экранах, корректируя стили по необходимости.

Практика 8: Создание динамических эффектов на Web-странице

1. **Изменение размеров рисунка:**
   - <img src="image.jpg" id="dynamicImage" alt="Dynamic Image">
   - #dynamicImage {
     width: 200px;
     transition: width 0.5s;
     }
   - const image = document.getElementById('dynamicImage');
     image.addEventListener('mouseover', () => {
     image.style.width = '300px';
     });
     image.addEventListener('mouseout', () => {
     image.style.width = '200px';
     });
2. **Замена слова на рисунок и обратно:**
   - Слово
     <img src="image.png" id="dynamicImage" alt="Image"
     style="display:none;">

- ```
  const text = document.getElementById('dynamicText');
  const img = document.getElementById('dynamicImage');

  text.addEventListener('click', () => {
    text.style.display = 'none';
    img.style.display = 'block';
  });

  img.addEventListener('click', () => {
    img.style.display = 'none';
    text.style.display = 'block';
  });
  ```

3. **Тестирование:**
   - Проверьте, как элементы реагируют на взаимодействие, и убедитесь в плавности эффектов.

Практика 9: Использование операций редактирования поверхностей 3D-объектов в Unity через 3ds MAX

1. **Создание 3D-объекта в 3ds MAX:**
   - Создайте объект, например, модель здания, добавив примитивы и применив модификаторы для детализирования.
2. **Экспорт модели:**
   - Экспортируйте объект в формате FBX:
     - `File` > `Export` > `Export Selected` > выберите `FBX`.
3. **Импорт в Unity:**
   - Откройте Unity и импортируйте FBX-файл в папку `Assets`.
4. **Редактирование поверхностей в Unity:**
   - В Unity выберите импортированный объект и добавьте материал с нужными текстурами.
   - Используйте инструменты редактирования поверхностей, такие как изменение нормалей или добавление деталей через скрипты.
5. **Тестирование:**
   - Разместите объект на сцене и проверьте его внешний вид и взаимодействие с освещением.
  ```

Практика 10: Использование кватернионов для программирования поворотов 3D-объектов в Unity

1. **Создание скрипта:**
    - В Unity создайте новый C# скрипт, например, `RotateWithQuaternion.cs`.
2. **using UnityEngine;**

    public class RotateWithQuaternion : MonoBehaviour
    {
       public Vector3 rotationAxis = Vector3.up;
       public float rotationSpeed = 90f;

       void Update()
       {
          Quaternion rotation = Quaternion.AngleAxis(rotationSpeed *
    Time.deltaTime, rotationAxis);
          transform.rotation = rotation * transform.rotation;
       }
    }
3. **Применение скрипта:**
    - Перетащите скрипт на 3D-объект в сцене.
4. **Настройка параметров:**
    - В инспекторе задайте ось вращения и скорость поворота.
5. **Тестирование:**
    - Запустите сцену и убедитесь, что объект вращается вокруг заданной оси с указанной скоростью.

Практика 11: Использование ключевых слов “Horizontal” и “Vertical” для программирования движения 3D-объектов с клавиатуры в Unity

1. **Создание скрипта:**
    - Создайте новый C# скрипт, например, `PlayerMovement.cs`.
2. **using UnityEngine;**

```
public class PlayerMovement : MonoBehaviour
{
    public float speed = 5f;

    void Update()
    {
        float moveHorizontal = Input.GetAxis("Horizontal");
        float moveVertical = Input.GetAxis("Vertical");

        Vector3 movement = new Vector3(moveHorizontal, 0.0f, moveVertical);
        transform.Translate(movement * speed * Time.deltaTime, Space.World);
    }
}
```

3. **Применение скрипта:**
    - Перетащите скрипт на 3D-объект, который будет управляться (например, персонаж).
4. **Настройка параметров:**
    - В инспекторе задайте скорость движения объекта.
5. **Тестирование:**
    - Запустите сцену и используйте клавиши WASD или стрелки для перемещения объекта по сцене.

Практика 12: Использование ключевых слов “Mouse X” и “Mouse Y” для программирования поворотов 3D-объектов мышью в Unity

1. **Создание скрипта:**
    - Создайте новый C# скрипт, например, `MouseRotation.cs`.
2. **using UnityEngine;**

```
public class MouseRotation : MonoBehaviour
{
    public float sensitivity = 100f;
    private float rotationX = 0f;
    private float rotationY = 0f;

    void Update()
    {
        float mouseX = Input.GetAxis("Mouse X") * sensitivity * Time.deltaTime;
```

```
        float mouseY = Input.GetAxis("Mouse Y") * sensitivity * Time.deltaTime;

        rotationX += mouseX;
        rotationY -= mouseY;
        rotationY = Mathf.Clamp(rotationY, -90f, 90f);

        transform.rotation = Quaternion.Euler(rotationY, rotationX, 0f);
    }
}
```

3. **Применение скрипта:**
    - Перетащите скрипт на 3D-объект, который необходимо вращать (например, камеру или персонажа).
4. **Настройка параметров:**
    - В инспекторе задайте чувствительность вращения.
5. **Тестирование:**
    - Запустите сцену и перемещайте мышь, чтобы вращать объект по горизонтали и вертикали.

Практика 13: Создание шаблонов Prefabs и программное создание экземпляров объектов из шаблона в Unity

1. **Создание Prefab:**
    - Создайте 3D-объект в сцене (например, куб).
    - Перетащите объект из сцены в папку `Assets` для создания Prefab.
2. **Создание скрипта для спавна объектов:**
    - Создайте новый C# скрипт, например, `PrefabSpawner.cs`.
3. **using UnityEngine;**

    public class PrefabSpawner : MonoBehaviour
    {
        public GameObject prefab;
        public int numberOfObjects = 10;
        public float spread = 5f;

        void Start()
        {
            for (int i = 0; i < numberOfObjects; i++)
            {

```
            Vector3 position = new Vector3(
                Random.Range(-spread, spread),
                0,
                Random.Range(-spread, spread)
            );
            Instantiate(prefab, position, Quaternion.identity);
        }
    }
}
```

4. **Применение скрипта:**
    - Создайте пустой GameObject в сцене и прикрепите к нему скрипт `PrefabSpawner`.
    - В инспекторе перетащите Prefab в поле `Prefab`.
5. **Настройка параметров:**
    - Установите количество объектов и диапазон их распределения.
6. **Тестирование:**
    - Запустите сцену и убедитесь, что экземпляры Prefab создаются в указанных позициях.

Практика 14: Использование триггеров на основе функций OnTriggerEnter() и OnTriggerExit() в Unity

1. **Настройка объектов:**
    - Создайте два 3D-объекта (например, сферу и куб).
    - На оба объекта добавьте компонент `Collider` и установите галочку `Is Trigger` для одного из них.
2. **Создание скрипта триггера:**
    - Создайте новый C# скрипт, например, `TriggerEvents.cs`.
3. **using UnityEngine;**

    public class TriggerEvents : MonoBehaviour
    {
        void OnTriggerEnter(Collider other)
        {
            Debug.Log("Объект вошел в триггер: " + other.name);
        }

        void OnTriggerExit(Collider other)

```
    {
        Debug.Log("Объект покинул триггер: " + other.name);
    }
}
```

4. **Применение скрипта:**
    - Перетащите скрипт на объект с настройкой `Is Trigger`.
5. **Тестирование:**
    - Переместите другой объект в область триггера и наблюдайте сообщения в консоли Unity.

Практика 15: Использование триггера на основе функции OnTriggerStay() в Unity

1. **Настройка объектов:**
    - Создайте два 3D-объекта (например, куб и сферу).
    - На один из объектов добавьте компонент `Collider` и установите галочку `Is Trigger`.
2. **Создание скрипта триггера:**
    - Создайте новый C# скрипт, например, `TriggerStay.cs`.
3. **using UnityEngine;**

    public class TriggerStay : MonoBehaviour
    {
        void OnTriggerStay(Collider other)
        {
            Debug.Log("Объект находится внутри триггера: " + other.name);
            // Дополнительные действия, например, изменение цвета объекта
            other.GetComponent<Renderer>().material.color = Color.red;
        }
    }
4. **Применение скрипта:**
    - Перетащите скрипт на объект с триггером.
5. **Тестирование:**
    - Поместите другой объект внутри триггера и наблюдайте изменения, например, изменение цвета или повторяющиеся сообщения в консоли.

Практика 16: Настройка 3D-звука и обработка различных звуков на сцене в Unity

1. **Добавление звуковых файлов:**
    - Импортируйте звуковые файлы в папку `Assets`.
2. **Создание источников звука:**
    - Создайте 3D-объекты, например, кубы, и добавьте к ним компонент `Audio Source`.
    - В настройках `Audio Source` выберите импортированный звуковой файл и установите параметр `Spatial Blend` на `3D`.
3. **Настройка звука:**
    - Настройте параметры звука, такие как `Volume`, `Pitch`, `Max Distance` и другие для реалистичного звучания.
4. **Создание скрипта для управления звуком:**
    - Создайте новый C# скрипт, например, `SoundController.cs`.
    - using UnityEngine;

      public class SoundController : MonoBehaviour
      {
        private AudioSource audioSource;

        void Start()
        {
          audioSource = GetComponent<AudioSource>();
        }

        void OnMouseDown()
        {
          audioSource.Play();
        }
      }
5. **Применение скрипта:**
    - Перетащите скрипт на объекты с `Audio Source`.
6. **Тестирование:**
    - Запустите сцену и щелкните по объектам, чтобы воспроизвести звуки.

Практика 17: Использование корутин для задержки выполнения действий в Unity

1. **Создание скрипта:**
    - Создайте новый C# скрипт, например, `CoroutineExample.cs`.
2. **using UnityEngine;**
   using System.Collections;

   public class CoroutineExample : MonoBehaviour
   {
   void Start()
   {
   StartCoroutine(DelayedAction());
   }

   IEnumerator DelayedAction()
   {
   Debug.Log("Действие начато");
   yield return new WaitForSeconds(3f);
   Debug.Log("Действие выполнено после 3 секунд");
   }
   }
3. **Применение скрипта:**
    - Перетащите скрипт на любой объект в сцене.
4. **Тестирование:**
    - Запустите сцену и наблюдайте за задержкой в выполнении действий, подтвержденной сообщениями в консоли.

Практика 18: Обработка щелчков мышью по 3D-объектам сцены в Unity

1. **Создание 3D-объекта:**
    - Создайте объект, например, куб, и добавьте к нему компонент `Collider`.
2. **Создание скрипта:**
    - Создайте новый C# скрипт, например, `ObjectClick.cs`.
3. **using UnityEngine;**

   public class ObjectClick : MonoBehaviour

```
{
    void OnMouseDown()
    {
        Debug.Log(gameObject.name + " был щелкнут");
        // Дополнительные действия, например, изменение цвета
        GetComponent<Renderer>().material.color = Color.green;
    }
}
```

4. **Применение скрипта:**
    - Перетащите скрипт на объект с `Collider`.
5. **Тестирование:**
    - Запустите сцену и щелкните по объекту, чтобы увидеть сообщение в консоли и изменения цвета.

Практика 19: Обработка щелчков мышью по кнопкам UI на CANVAS в Unity

1. **Создание UI-кнопки:**
    - В панели `Hierarchy` выберите `UI` > `Button` для добавления кнопки на CANVAS.
2. **Создание скрипта:**
    - Создайте новый C# скрипт, например, `UIButtonHandler.cs`.
3. **Кодирование обработки щелчка:**

```
Plain Text
using UnityEngine;
using UnityEngine.UI;

public class UIButtonHandler : MonoBehaviour
{
    public Button myButton;

    void Start()
    {
        myButton.onClick.AddListener(OnButtonClick);
    }

    void OnButtonClick()
    {
        Debug.Log("Кнопка была нажата");
        // Дополнительные действия, например, изменение текста
        myButton.GetComponentInChildren<Text>().text = "Нажато";
```

```
    }
}
```

4. **Применение скрипта:**
    - Перетащите скрипт на объект CANVAS или отдельный пустой GameObject.
    - В инспекторе назначьте кнопку `myButton`.
5. **Тестирование:**
    - Запустите сцену и нажмите на кнопку, чтобы увидеть сообщение в консоли и изменение текста.

Практика 20: Обработка объекта при наведении и уходе курсора мыши с кнопки на CANVAS в Unity

1. **Создание UI-кнопки:**
    - В панели `Hierarchy` выберите `UI` > `Button` для добавления кнопки на CANVAS.
2. **Создание скрипта:**
    - Создайте новый C# скрипт, например, `UIButtonHover.cs`.
3. **using UnityEngine;**
   using UnityEngine.EventSystems;
   using UnityEngine.UI;

   public class UIButtonHover : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler
   {
   public Image buttonImage;

   public void OnPointerEnter(PointerEventData eventData)
   {
   buttonImage.color = Color.yellow;
   }

   public void OnPointerExit(PointerEventData eventData)
   {
   buttonImage.color = Color.white;
   }
   }
4. **Применение скрипта:**

- Перетащите скрипт на кнопку.
- В инспекторе назначьте `buttonImage` на компонент `Image` кнопки.

5. **Тестирование:**
    - Запустите сцену и наведите курсор на кнопку, чтобы увидеть изменение цвета, а затем уберите курсор и верните цвет к исходному.

1. Создать в среде Adobe Animate анимацию перемещения объекта по заданной траектории и программу на языке ActionScript 3.0 для запуска и остановки ее воспроизведения кнопками

Шаги выполнения

1. **Подготовка сцены и объекта**
    - Создайте новый файл в Adobe Animate (Flash ActionScript 3.0).
    - Нарисуйте (или импортируйте) объект, который будет перемещаться.
    - Преобразуйте объект в символ (F8), назовите его, например, `movingObject` и выберите тип символа *MovieClip*.
2. **Создание motion tween и траектории движения**
    - На **слое 1** разместите ваш объект на первом кадре.
    - Щелкните правой кнопкой мыши на кадре 1 и выберите «Создать анимацию движения».
    - Переместите ползунок на шкале времени (например, на кадр 50).
    - Измените положение объекта на сцене, чтобы автоматически создалась **траектория движения** (см. [1][3][5]).
    - Настройте саму траекторию, используя инструменты **Выделение** или **Спецвыделение**, если необходимо изменить форму пути (см. [3][5]).
3. **Добавление кнопок**
    - Создайте на сцене два прямоугольника (или импортируйте готовые изображения), которые будут кнопками «Play» и «Stop».
    - Преобразуйте каждый прямоугольник в символ типа *Button*.
    - Назначьте экземплярам кнопок имена (например, `playBtn` и `stopBtn`).

4. **ActionScript 3.0 для запуска и остановки**
    - stop(); // Останавливаем анимацию на 1 кадре

      playBtn.addEventListener(MouseEvent.CLICK, onPlayClick);
      stopBtn.addEventListener(MouseEvent.CLICK, onStopClick);

      function onPlayClick(e:MouseEvent):void {
        play(); // Запускаем воспроизведение со следующего кадра
      }

      function onStopClick(e:MouseEvent):void {
        stop(); // Останавливаем анимацию на текущем кадре
      }
    - Убедитесь, что вы присвоили правильные *имена экземпляров* кнопкам в панели **Properties** (в поле *Instance Name*).
5. **Тестирование**
    - Нажмите **Control + Enter** для проверки.
    - При нажатии на кнопку «Play» анимация запускается, при нажатии на «Stop» — останавливается.

2. Создать в среде Adobe Animate анимацию перемещения объекта по заданной траектории и программу на языке JavaScript для запуска и остановки ее воспроизведения кнопками

Шаги выполнения (HTML5 Canvas)

1. **Создание проекта**
    - Создайте новый файл типа **HTML5 Canvas** в Adobe Animate.
    - Нарисуйте или импортируйте объект, преобразуйте его в символ типа *MovieClip* (аналогично пункту 1 из предыдущего задания).
2. **Анимация движения по траектории**
    - На временной шкале примените **Classic Tween** или **Motion Tween** (в зависимости от версии) для перемещения объекта по заданной кривой (см. [1][3][5]).
    - Убедитесь, что объект имеет *имя экземпляра*, например, `movingObject`.
3. **Создание кнопок**

- Создайте два графических элемента для кнопок, преобразуйте их в символы типа *Button* либо *MovieClip*.
- Назовите их, например, `playBtn` и `stopBtn`, присвоив соответствующие *имена экземпляров*.

4. **Код на JavaScript (CreateJS)**
   - this.stop();

     this.playBtn.addEventListener("click", playAnimation.bind(this));
     this.stopBtn.addEventListener("click", stopAnimation.bind(this));

     function playAnimation() {
      this.play();
     }

     function stopAnimation() {
      this.stop();
     }
   - Данный код использует библиотеку CreateJS, автоматически подключаемую в **HTML5 Canvas** проектах Animate (см. [7], раздел про AS3, но логика в JS-среде аналогична).
5. **Тестирование**
   - Опубликуйте проект (File → Publish) или нажмите **Control + Enter**.
   - Убедитесь в корректной работе кнопок «Play» и «Stop».

3. Создать в среде Unity программу для обхода камерой вокруг центра объекта на сцене с помощью мыши

Шаги выполнения

1. **Подготовка сцены**
   - Создайте 3D-объект, например, `Cube`, в центре сцены (координаты (0, 0, 0)).
   - Добавьте `Main Camera` на сцену, расположив её, например, на некотором удалении по оси Z.
2. **Скрипт для вращения камеры вокруг объекта**
   - using UnityEngine;

     public cla`ss CameraOrbit` : MonoBehaviour
     {

```
    public Transform target;        // Цель, вокруг которой
вращаемся
    public float distance = 5.0f;   // Расстояние от цели
    public float xSpeed = 120.0f;   // Скорость вращения по горизонтали
    public float ySpeed = 120.0f;   // Скорость вращения по вертикали

    private float x = 0.0f;
    private float y = 0.0f;

    void Start()
    {
        Vector3 angles = transform.eulerAngles;
        x = angles.y;
        y = angles.x;

        if (target == null)
        {
            GameObject go = GameObject.Find("Cube");
            if (go != null) target = go.transform;
        }
    }

    void LateUpdate()
    {
        if (target)
        {
            if (Input.GetMouseButton(0)) // Зажать левую кнопку мыши
            {
                x += Input.GetAxis("Mouse X") * xSpeed * 0.02f;
                y -= Input.GetAxis("Mouse Y") * ySpeed * 0.02f;
                y = Mathf.Clamp(y, -80, 80);
            }

            Quaternion rotation = Quaternion.Euler(y, x, 0);
            Vector3 position = rotation * new Vector3(0.0f, 0.0f, -distance) +
target.position;

            transform.rotation = rotation;
            transform.position = position;
        }
    }
}
```

3. **Настройка инспектора**

- В поле `Target` перетащите ваш `Cube` (или любой другой объект), вокруг которого нужно вращаться.
- Настройте расстояние (`distance`) и скорости вращения (`xSpeed`, `ySpeed`) по своему вкусу.

4. **Тестирование**
    - Запустите сцену и зажмите **левую кнопку мыши**, двигая курсор, чтобы поворачивать камеру вокруг объекта.

4. Создать в среде Unity программу выбора оптимального ракурса размещения и поворота камеры для просмотра объекта на сцене щелчком мышью по кнопке на CANVAS

Шаги выполнения

1. **Добавление UI-кнопки**
    - В меню «GameObject → UI → Button» создайте кнопку.
    - Добавьте на сцену `EventSystem` (если не создался автоматически).
2. **Скрипт для переключения вида**
    - using UnityEngin`e;`
      `using UnityEn`gine.UI;

      public class SetCameraView : MonoBehav`iour`
      `{`
      public `Camera` mainCamera;
      public Transform customViewTransform; // Позиция и поворот «оптимального ракурса»

      public Button viewButton;

      void Start()
      {
      viewButton.onClick.AddListener(SetOptimalView);
      }

      void SetOptimalView()
      {
      if (mainCamera != null && customViewTransform != null)

```
        {
            mainCamera.transform.position =
    customViewTransform.position;
            mainCamera.transform.rotation = customViewTransform.rotation;
        }
    }
}
```

- `customViewTransform` — это пустой объект, который вы можете разместить в сцене в «оптимальной» точке обзора, а затем в инспекторе перетащить его в поле.

3. **Настройка инспектора**
    - Перетащите `Main Camera` в поле `mainCamera`.
    - Создайте пустой объект (например, `OptimalView`), разместите его в нужном месте (камера там, где вы хотите). Перетащите его в `customViewTransform`.
    - Перетащите саму кнопку UI в поле `viewButton`.
4. **Тестирование**
    - Запустите сцену. Изначально камера может быть в любой позиции.
    - Нажмите на кнопку, и камера должна переместиться и повернуться к «оптимальному ракурсу».

5. Создать в среде Unity программу поступательного движения и вращения для объектов на сцене

Шаги выполнения

1. **Настройка сцены**
    - Разместите 3D-объекты на сцене (кубы, сферы и т.д.).
2. **Скрипт движения и вращения**
    - ```
      using UnityEngine;

      public class MoveAndRotate : MonoBehaviour
      {
          public Vector3 moveSpeed = new Vector3(1, 0, 0);
          public Vector3 rotateSpeed = new Vector3(0, 90, 0);

          void Update()
          {
              // Поступательное движение
      ```
      ```

```
        transform.Translate(moveSpeed * Time.deltaTime, Space.World);

        // Вращение
        transform.Rotate(rotateSpeed * Time.deltaTime, Space.Self);
    }
}
```

3. **Применение**
    - Навесьте скрипт на любой объект, настройте скорости движения и вращения.
4. **Проверка**
    - Запустите сцену и наблюдайте за движением и вращением объекта.

6. Создать в среде Unity программу вращения 3D-объекта с помощью кватернионов Quaternion вокруг произвольной оси

Шаги выполнения

1. **Скрипт вращения**
    - using UnityEngin`e;`

      `public class Quat`ernionRotation : MonoBehaviour
      {
        public Vector3 axis = Vector3.up; // Ось вращения
        public float speed = 90f;   // Градусы в секунду

        void Update()
        {
          // Угол поворота за кадр
          float angle = speed * Time.deltaTime;
          Quaternion rotation = Quaternion.AngleAxis(angle, axis);
          transform.rotation = rotation * transform.rotation;
        }
      }
2. **Настройка**
    - Навесьте скрипт на нужный 3D-объект, выберите ось (например, `(1, 1, 0)`) и задайте скорость вращения.
3. **Тестирование**

- Запустите сцену и проверьте корректность вращения вокруг заданной оси (см. общий подход к кватернионам также в предыдущих примерах).

7. Создать в среде Unity программу генерации объекта на сцене из префаба в случайной позиции на сцене при нажатии клавиши клавиатуры

Шаги выполнения

1. **Подготовка префаба**
   - Создайте в сцене объект, настройте его компоненты.
   - Перетащите объект из Hierarchy в папку **Assets** в Project, чтобы создать префаб (см. [2][4][8][9]).
2. **Скрипт генерации**
   - using UnityEngin`e;`

     ```
     public class PrefabSpawner : MonoBehaviour
     {
        public GameObject prefab;
        public float range = 5f; // Диапазон случайных координат

        void Update()
        {
           if (Input.GetKeyDown(KeyCode.Space))
           {
              Vector3 randomPos = new Vector3(
                 Random.Range(-range, range),
                 0,
                 Random.Range(-range, range)
              );
              Instantiate(prefab, randomPos, Quaternion.identity);
           }
        }
     }
     ```
3. **Настройка**
   - Перетащите ваш префаб в поле `prefab` в инспекторе.
   - Установите желаемое значение `range`.
4. **Тестирование**

- Запустите сцену и нажмите клавишу **Пробел** (Space). Каждый раз будет создаваться новый экземпляр префаба в случайной позиции.

8. Создать в среде Unity программу для смены цвета 3D-объекта на сцене при щелчке по нему мышью

Шаги выполнения

1. **3D-объект и коллайдер**
   - Добавьте на объект компонент `Collider` (BoxCollider, SphereCollider и т.д.).
2. **Скрипт**
   - using Uni`tyEngine;`

     `publ` ic class ColorOnClick : MonoBehaviour
     {
     private Renderer rend;

     void Start()
     {
     rend = GetComponent<Renderer>();
     }

     void OnMouseDown()
     {
     rend.material.color = Random.ColorHSV(); // Случайный цвет
     }
     }
3. **Применение**
   - Повесьте скрипт на ваш объект.
4. **Тест**
   - Запустите сцену, кликните на объект — его цвет должен смениться (см. аналогичную логику в заданиях про `OnMouseDown`).

9. Создать в среде Unity программу вращения 3D-объекта клавишами клавиатуры

Шаги выполнения

1. **Скрипт**
    - using Uni`tyEngine;`

      ```
      public class RotateWithKeys : MonoBehaviour
      {
          public float rotationSpeed = 90f;

          void Update()
          {
              if (Input.GetKey(KeyCode.LeftArrow))
              {
                  transform.Rotate(Vector3.up, -rotationSpeed * Time.deltaTime);
              }
              if (Input.GetKey(KeyCode.RightArrow))
              {
                  transform.Rotate(Vector3.up, rotationSpeed * Time.deltaTime);
              }
              if (Input.GetKey(KeyCode.UpArrow))
              {
                  transform.Rotate(Vector3.right, -rotationSpeed *
      Time.deltaTime);
              }
              if (Input.GetKey(KeyCode.DownArrow))
              {
                  transform.Rotate(Vector3.right, rotationSpeed * Time.deltaTime);
              }
          }
      }
      ```

2. **Настройка**
    - Добавьте скрипт на ваш объект.
3. **Проверка**
    - При запуске сцены используйте стрелки ←→↑↓ для вращения объекта.

10. Создать в среде Unity программу вращения 3D-объекта с помощью мыши

Шаги выполнения

1. **Скрипт**
    - using Uni`tyEngine;`

```
pub lic class MouseRotate : MonoBehaviour
{
    public float mouseSensitivity = 100f;
    private float rotationY = 0f;
    private float rotationX = 0f;

    void Update()
    {
        float mouseX = Input.GetAxis("Mouse X") * mouseSensitivity *
Time.deltaTime;
        float mouseY = Input.GetAxis("Mouse Y") * mouseSensitivity *
Time.deltaTime;

        rotationX += mouseX;
        rotationY -= mouseY;
        rotationY = Mathf.Clamp(rotationY, -80f, 80f);

        transform.rotation = Quaternion.Euler(rotationY, rotationX, 0f);
    }
}
```

2. **Применение**
    - Повесьте скрипт на ваш 3D-объект (или на камеру, если вращаете камеру).
3. **Тестирование**
    - Запустите сцену и перемещайте мышь, чтобы вращать объект.

11. Создать в среде Unity программу для обработки столкновения двух 3D-объектов с изменением их цвета

Шаги выполнения

1. **Настройка физики**
    - У обоих объектов должны быть коллайдеры (например, `BoxCollider`) и по крайней мере один объект должен иметь `Rigidbody`.
2. **Скрипт**
    - using UnityEngine;

      `public class CollisionC` olorChange : MonoBehaviour
      {

```
    private Renderer rend;

    void Start()
    {
        rend = GetComponent<Renderer>();
    }

    void OnCollisionEnter(Collision collision)
    {
        rend.material.color = Color.red;
    }

    void OnCollisionExit(Collision collision)
    {
        rend.material.color = Color.white;
    }
}
```

3. **Тестирование**
    - При столкновении объект меняет цвет на красный, при выходе из столкновения — возвращается к белому.

12. Создать в среде Unity программу для обработки входа и выхода 3D-объекта в триггер с изменением цвета объекта

Шаги выполнения

1. **Коллайдер с IsTrigger**
    - Например, создайте `Cube` и поставьте галочку `Is Trigger` в компоненте `BoxCollider`.
2. **Скрипт**
    - using Uni`tyEngine;`

      `public cla`ss TriggerColorChange : MonoBehaviour
      {
          void OnTriggerEnter(Collider other)
          {
              other.GetComponent<Renderer>().material.color = Color.green;
          }

          void OnTriggerExit(Collider other)
          {

```
            other.GetComponent<Renderer>().material.color = Color.white;
        }
    }
```

3. **Тест**
    - Другой объект с `Rigidbody` при входе в триггер станет зеленым, а при выходе вернется в белый цвет.

13. Создать в среде Unity программу для озвучивания момента столкновения 3D-объектов на сцене

Шаги выполнения

1. **Импорт звукового файла**
    - Перетащите звуковой файл (например, `collision.wav`) в `Assets`.
2. **Скрипт**
    - using Uni`tyEngine;`

      `public` class Colli`sionSound :` MonoBehaviour
      ```
      {
          private AudioSource audioSource;

          void Start()
          {
              audioSource = GetComponent<AudioSource>();
          }

          void OnCollisionEnter(Collision collision)
          {
              if (!audioSource.isPlaying)
              {
                  audioSource.Play();
              }
          }
      }
      ```
3. **Настройка**
    - Поместите аудиоклип в `AudioSource`.
    - Убедитесь, что у объектов настроены коллайдеры и `Rigidbody`.
4. **Проверка**

- Запустите сцену и столкните объекты — при ударе должен проигрываться звук.

14. Создать в среде Unity анимацию перемещения объекта на сцене и программу для ее запуска при щелчке по кнопке CANVAS

Шаги выполнения

1. **Анимация**
   - Откройте `Window → Animation → Animation`.
   - Выделите объект, нажмите «Create», чтобы создать анимационный клип (например, `MoveAnim`).
   - Запишите ключи для положения объекта (например, перемещение по оси X).
2. **Animator Controller**
   - Unity автоматически создаст `Animator Controller`. Убедитесь, что анимация связана с объектом.
3. **UI-кнопка**
   - Создайте кнопку на сцене.
4. **Скрипт**
   - using Uni`tyEngine;`
     `using UnityEn`gine.UI;

     public class PlayAnimationOnClick : MonoBehaviour
     {
     public Animator animator;
     public Button playButton;
     public string animationTriggerName = "PlayMove";

     void Start()
     {
     playButton.onClick.AddListener(OnPlayClicked);
     }

     void OnPlayClicked()
     {
     animator.SetTrigger(animationTriggerName);

```
    }
}
```

- В `Animator` создайте параметр *Trigger* `PlayMove` и свяжите переход в анимацию перемещения с этим триггером.

5. **Тест**
    - При нажатии на кнопку скрипт запускает триггер, и объект начинает воспроизводить анимацию.

15. Создать в среде Unity программу подсветки объекта на сцене при наведении курсора мыши на кнопку CANVAS

Шаги выполнения

1. **UI-кнопка**
    - Создайте кнопку на сцене.
2. **Скрипт**
    - using Uni`tyEngine;`
      `using UnityEn`gine.UI;
      using UnityEngine.EventSystems;

      public class ButtonHoverHighlight : MonoBehaviour,
      IPointerEnterHandler, IPointerExitHandler
      {
      public Renderer targetRenderer;

      public void OnPointerEnter(PointerEventData eventData)
      {
      if (targetRenderer != null)
      targetRenderer.material.color = Color.yellow;
      }

      public void OnPointerExit(PointerEventData eventData)
      {
      if (targetRenderer != null)
      targetRenderer.material.color = Color.white;
      }
      }
3. **Настройка**
    - На кнопку добавьте данный скрипт.

- Укажите в `targetRenderer` тот объект, который нужно «подсвечивать».

4. **Проверка**
    - При наведении курсора на кнопку (в интерфейсе), объект меняет цвет и возвращается к исходному при уходе курсора.

16. Создать в среде Unity программу появления на экране текстового окна при щелчке по кнопке на CANVAS

Шаги выполнения

1. **UI-элементы**
    - Создайте новый `Panel` или `Text` на Canvas и изначально сделайте его неактивным (`SetActive(false)`).
2. **Скрипт**
    - `using UnityEngin`e;
      using UnityEngine.UI;

      public class ShowTextPanel : MonoBehaviour
      {
      public GameObject textPanel;
      public Button showButton;

      void Start()
      {
      textPanel.SetActive(false);
      showButton.onClick.AddListener(OnShowClicked);
      }

      void OnShowClicked()
      {
      textPanel.SetActive(true);
      }
      }
3. **Настройка**
    - В инспекторе перетащите в `textPanel` ваш объект панель/текст.
    - В `showButton` — кнопку.
4. **Проверка**
    - При щелчке по кнопке окно становится видимым.

17. Создать в среде Unity программу обработки щелчков мышью по 3D-объектам сцены

Шаги выполнения

1. **Объекты и коллайдеры**
   - Для каждого 3D-объекта добавьте компонент `Collider`.
2. **Скрипт**
   - public class ClickableObject : MonoBehaviour
     {
     void OnMouseDown()
     {
     Debug.Log("Object clicked: " + gameObject.name);
     }
     }
3. **Проверка**
   - Запустите, кликните по объекту, сообщение появится в консоли Unity.

18. Создать на Web-странице функцию формирования блока диалогового окна за пределами страницы с вложенным блоком для кнопки запуска функции анимации выдвижения/скрытия окна на центр страницы

Шаги выполнения

1. **<button id="o**penDialogBtn">Открыть диалог</button>

   <button id="closeDialogBtn">Скрыть</button>

2. **.slide-down {**
   animation: slideDown 1s forwards;
   }

   .slide-up {

```
    animation: slideUp 1s forwards;
}

@keyframes slideDown {
    from { top: -200px; display: block; }
    to { top: 50px; display: block; }
}

@keyframes slideUp {
    from { top: 50px; }
    to { top: -200px; }
}
```

3. **const open**Btn = document.getElementById('openDialogBtn');
   const closeBtn = document.getElementById('closeDialogBtn');
   const dialog = document.getElementById('dialog');

   openBtn.addEventListener('click', () => {
   dialog.style.display = 'block';
   dialog.classList.remove('slide-up');
   dialog.classList.add('slide-down');
   });

   closeBtn.addEventListener('click', () => {
   dialog.classList.remove('slide-down');
   dialog.classList.add('slide-up');
   setTimeout(() => {
   dialog.style.display = 'none';
   }, 1000);
   });
4. **Проверка**
   - При нажатии «Открыть диалог» окно плавно выезжает, при «Скрыть» — уезжает вверх и скрывается.

19. Создать на Web-странице функцию формирования на центре страницы блока диалогового окна с вложенными блоками для вывода ответа и блоком заголовка над ним

Шаги выполнения (пример)

1. **<div** id="dialogBox" style="position: fixed; top: 50%; left: 50%; transform: translate(-50%, -50%); width: 400px; padding: 20px; border: 1px solid #000; background: #fff; display: none;">
   <h2 id="dialogTitle">Заголовок</h2>
   Здесь будет вывод ответа

<button id="showDialogBtn">Показать диалог</button>
2. **#dialogBox {**
   box-shadow: 0 0 10px rgba(0,0,0,0.5);
   z-index: 999;
}
3. **const show**DialogBtn = document.getElementById('showDialogBtn');
const dialogBox = document.getElementById('dialogBox');

showDialogBtn.addEventListener('click', () => {
   dialogBox.style.display = 'block';
   document.getElementById('dialogTitle').textContent = 'Подписка оформлена';
   document.getElementById('dialogContent').textContent = 'Спасибо за подписку!';
});
4. **Результат**
   - При нажатии на кнопку появляется окно в центре, с заголовком и ответом.

20. Создать на Web-странице функцию формирования на центре страницы блока диалогового окна с блоком полосы для ввода вопроса и кнопкой, запускающей функцию вывода ответа

Шаги выполнения

1. **<div** id="questionDialog" style="position: fixed; top: 50%; left: 50%; transform: translate(-50%, -50%); width: 400px; padding: 20px; background: #fafafa; border: 1px solid #ccc; display: none;">
   <h3>Введите вопрос:</h3>
   <input type="text" id="questionInput" />
   <button id="askBtn">Спросить</button>

```
</div>

<button id="openQuestionDialogBtn">Задать вопрос</button>
```

2. **const open**QuestionBtn =
```
document.getElementById('openQuestionDialogBtn');
const questionDialog = document.getElementById('questionDialog');
const askBtn = document.getElementById('askBtn');
const answerOutput = document.getElementById('answerOutput');

openQuestionBtn.addEventListener('click', () => {
    questionDialog.style.display = 'block';
});

askBtn.addEventListener('click', () => {
    const question = document.getElementById('questionInput').value;
    // Простейшая логика вывода ответа
    if (question.trim() !== "") {
        answerOutput.textContent = "Ответ на ваш вопрос: " +
question.toUpperCase() + "!";
    } else {
        answerOutput.textContent = "Вы не ввели вопрос.";
    }
});
```

3. **Пояснения**
    - При нажатии на «Задать вопрос» появляется диалоговое окно с полем ввода.
    - При нажатии «Спросить» выводится «ответ».
