***Сорокин М.М.***

***Научный руководитель: ассистент каф. МОП ЭВМ Гуляев Н.А.***

Южный федеральный университет, ИКТИБ, Таганрог

mikhail.fourtyn@gmail.com

**ОБ ОПЫТЕ РЕАЛИЗАЦИИ ВИЗУАЛЬНЫХ ЭФФЕКТОВ ПРИ РАЗРАБОТКЕ ИНТЕРАКТИВНЫХ ГРАФИЧЕСКИХ ПРИЛОЖЕНИЙ НА UNREAL ENGINE И UNITY**

*Рассмотрен вопрос дизайна и технической реализации визуальных эффектов в интерактивных графических приложениях. Рассмотрены аспекты реализации визуальных эффектов, на базе Unreal Engine 4 и Unity, произведено сравнение нюансов дизайна и программной реализации.*

*РАЗРАБОТКА ПРОГРАММНЫХ ПРИЛОЖЕНИЙ, ИНТЕРАКТИВНЫЕ ГРАФИЧЕСКИЕ ПРИЛОЖЕНИЯ, ИГРОВОЙ ДВИЖОК, UNREAL ENGINE, UNITY, ВИЗУАЛЬНЫЕ ЭФФЕКТЫ.*

Визуальные эффекты (VFX) в современных интерактивных графических приложениях и игровых приложениях – опциональная составляющая виртуальной сцены, однако применение визуальных эффектов существенно дополняет стандартные средства визуализации, например, за счёт комбинации различных техник рендеринга, что позволяет подчеркнуть какие-либо значимые элементы сцены, сформировать убедительный, реалистичный образ визуализируемых объектов и явлений, а также значительно улучшить пользовательский опыт работы с приложением или игрой.

С технической точки зрения визуальные эффекты могут быть реализованы различными способами, однако, как правило, визуальный эффект формируется посредством дополнения (например, наложение поверх за счёт дополнительной итерации рендеринга) уже существующей сцены некоторыми двумерными или трёхмерными объектами. Таким образом, реализация визуального эффекта в большинстве случаев подразумевает наличие нескольких составляющих: плоская (двухмерная) фигура или полигональная сетка, текстура и шейдер. Например, в случае реализации системы частиц (часто используется в современных игровых приложениях), частицы, имеющие форму круга, фактически состоят из полигона (который, в свою очередь, представляет собой два треугольника) и материала, определяющего совокупность различных ресурсов, таких как текстура, конкретный шейдер и так далее. Кроме того, необходима система управления частицами, которая будет управлять созданием, процессом жизни и удалением со сцены набора разнообразных частиц, составляющих собой визуальный эффект.

Но, прежде чем перейти к более детальному рассмотрению аспектов технической реализации визуальных эффектов, следует рассмотреть другой не менее важный аспект – аспект дизайна. Главный нюанс заключается в том, что общее восприятие эффекта определяется совокупностью множества факторов: цвет, форма, контрастность эффекта, время его подготовки, наличие и время существования оставленного им следа, что во многом роднит задачу визуализации эффектов с задачей построения интерфейса. По аналогии с элементами интерфейса, параметры визуализации эффектов необходимо «подгонять» под внутреннюю логику приложения в зависимости от назначения того или иного спецэффекта, например, если необходимо привлечь внимание пользователя, то визуально данный эффект должен быть ярче, контрастнее и так далее [1]. Например, в определённых жанрах игровых приложений спецэффекты (системы частиц и др.) задействуются для индикации выполнения игровыми персонажами тех или иных действий, применения тех или иных способностей. Так называемые «ультимативные способности» (обычно они требуют некой длительной подзарядки во время игры и доступны очень ограниченное количество раз за игровую сессию) могут быть выделены более интенсивными спецэффектами, нежели другие способности, для того чтобы сфокусировать внимание – чтобы игрок имел возможность среагировать и предпринять какие-либо действия, а также, чтобы визуально отразить большее влияние на игровой процесс.

Непосредственно сам визуальный эффект может быть связан с набором других, свойства которых могут иметь зависимость от свойств «родительского» эффекта или объекта-носителя, например, при визуализации эффекта летящего огненного снаряда, сам снаряд делается ярким (за счёт специального шейдера или посредством реализации в качестве спецэффекта), а разлетающиеся от него частицы и след становятся тем более прозрачными и темными, чем дальше они оказываются от снаряда. Также важна и «цветовая тема» – в игровых приложениях ряда жанров игроки или персонажи разделяются на команды, где цветовое оформление эффектов персонажей связывается с соответствующей темой.

Цветовые темы могут иметь и более глобальное применение – так, например, в настоящее время подавляющее большинство игровых приложений следуют некоторой формально не зафиксированной, но действующей (поскольку привычна многим пользователям) схеме распределения цветовых оттенков. Например, сформировался стереотип относительно цветовых оттенков для визуализации радиации, радиоактивных объектов и субстанций (светло-зелёные оттенки, возможно, имеющие свечение), аналогичным образом был сформирован стереотип относительно «цвета» воздуха, ветра и иных связанных предметов и явлений (светло-синие оттенки).

С точки зрения реализации в конкретной среде, то есть в **Unreal Engine 4**, система **Niagara** (являющаяся бесплатным подключаемым к Unreal Engine 4 модулем) предоставляет разнообразный, простой в освоении и удобный в использовании функционал. Так, для создания группы частиц с общим поведением требуется создать **Niagara Emitter**. Например, для создания эффекта удара предмета о металлическую поверхность используется одна частица-вспышка и несколько однотипных частиц-искр. У них различное поведение, поэтому для них требуется два эмиттера (источника). Эмиттер состоит из набора модулей, организованных по принципу «стека модулей» (Module Stack). Каждый модуль реализует элемент функционала (например, добавляет частицам скорость или управляет изменением цвета частицы с течением времени). Непосредственно в редакторе предоставляется удобный функционал управления числовыми значениями полей (например, вектора или коэффициенты из одного числа) в модулях: может быть считан вектор из пользовательского параметра, установлено автоматическое вычисление вектора на основе задаваемого числа с плавающей запятой, автоматическое вычисление вектора по задаваемой кривой, или получение случайного вектора, заключенного в задаваемых пределах, и т. п.

На виртуальной сцене используются **Niagara System**, являющиеся, по сути, контейнерами для Niagara Emitter. Они объединяют эмиттеры в единое целое и позволяют использовать общие пользовательские параметры, доступные для редактирования из редактора сцен [2].

С другой стороны, среда и средство разработки интерактивных графических приложений **Unity** во многих аспектах является аналогом и прямым конкурентом [3] **Unreal Engine**, что может наблюдаться и в сфере обеспечения средствами реализации визуальных эффектов. Так, подключаемый к игровому движку Unity, бесплатный модуль **Particle System** предоставляет аналогичный, во многом совпадающий функционал. Одноименный компонент игрового объекта GameObject несет функционал, аналогичный Niagara Emitter в Unreal Engine. Единственным существенным отличием от описанной выше структуры Niagara System является то, что объект с компонентом Particle System уже сам по себе является игровым объектом, который можно расположить на игровой сцене, в отличие от Niagara Emitter, который не может существовать на сцене, не будучи частью Niagara System. Поэтому следует рассмотреть ключевые отличия в деталях реализации систем частиц в движках Unreal Engine 4 и Unity.

Например, в Unity даётся возможность создавать дочернюю систему частиц по одному из событий из предлагаемого списка, например, создании или удалении родительской. При этом производится выборочное наследование параметров родительской системы частиц в дочернюю (например, размера, цвета или времени жизни). В Unreal Engine также реализовано наследование систем частиц, но нет прямой привязки к упомянутым выше событиям.

Кроме того, в функционале Unity предлагается компонент Trail Renderer, который представляет собой прямоугольник, длина которого динамически изменяется пропорционально скорости объекта, к которому он привязан. Данный компонент применим, например, для создания следа, например, при визуализации инверсионного следа ракеты или самолёта.

В обоих движках предлагается графический редактор шейдеров с помощью визуального программирования. Шейдер представляется в виде графа, а также в обоих движках присутствует достаточно большое количество различных узлов, хотя в Unreal Engine узлы и настройки шейдеров более разнообразны.

Таким образом, различие между реализацией систем визуальных эффектов в двух движках в основном заключается в деталях, но именно эти детали могут сыграть определяющую роль в конкретных ситуациях, возникающих в ходе разработки конкретных проектов.

С точки зрения субъективного опыта разработки одинаковых сцен и реализации визуальных эффектов в обеих средах (в Unreal Engine 4 и в Unity), можно отметить, что в рассматриваемых движках могут быть созданы полностью аналогичные друг другу визуальные эффекты, при условии использования одних и тех же ресурсов (текстур и др.). Также возможно отметить, что трудозатраты и временные затраты для разработки эффектов в обеих средах сопоставимы, однако, разработка в Unreal Engine 4 представляется более структурированной и последовательно организованной с точки зрения дизайнера и разработчика.

**Библиографический список:**

1. Купер, А. Об интерфейсе. Основы проектирования взаимодействия / А. Купер, Р. Рейман, Д. Кронин. – СПб.: Символ-Плюс, 2009. – 649 с;

2. Niagara Overview // Unreal Engine Documentation URL: https://docs.unrealengine.com/en-US/Engine/Niagara/Overview/index.html (дата обращения: 24.11.2020);

3. Гуляев Н. А., Сорокин М. М., Об опыте разработки интерактивного графического приложения при помощи Unity3d // Информационные технологии, системный анализ и управление (ИТСАУ). – 2020. – С. 152-155.