ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Доцент департамента
программной инженерии
факультета компьютерных наук
канд. техн. наук

_____А. Р. Закиевна
«___» _____ 2016 г.

УТВЕРЖДАЮ
Академический руководитель
образовательной программы
«Программная инженерия»

_____ В. В. Шилов
«___» _____ 2016 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

ПРОГРАММА СКЕЛЕТНАЯ АНИМАЦИЯ

Техническое задание

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.509000-01 ТЗ 01-1-ЛУ

Исполнитель
студент группы 151 ПИ
_____ /А. М. Абрамов /
«____»_____ 2016 г.

2016

ПРОГРАММА СКЕЛЕТНАЯ АНИМАЦИЯ

Техническое задание

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.500900 ТЗ 01-1-ЛУ

Листов 16

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Содержание

1. Введение

1.1. Наименование

Наименование: «Программа скелетная анимация»

1.2. Краткая характеристика

Программа предназначена для быстрого просмотра анимационных файлов созданных в пакетах для 3-х мерного моделирования.

1.3. Документы, на основании которых ведется разработка

Приказ НИУ ВШЭ No 6.18.1-02/1112-19 от 11.12.2015 в соответствии с учебным планом подготовки бакалавров по направлению 09.03.04 «Программная инженерия».

2. Назначение разработки

2.1. Функциональное назначение

Функциональным назначением приложения является предоставление пользователю возможности быстро загрузить 3D анимацию из файла, проиграть ее, показывать анимацию учитывая связанные с ней материалы и нормали, просмотреть мешы и кости входящие в состав анимации, перейти к любому моменту времени в анимационном файле.

2.2. Эксплутационное назначение

Программа наглядно демонстрирует содержание файла экпортированного из пакетов для 3-х мерного моделированния. Она может использоваться в процессе отладки приложений использующих анимацию и в работе дизайнера 3D моделей. В силу простоты интерфейса она подходит для использования людям не очень хорошо знакомым с более мощными и трудными в использовании пакетами для 3-х мерного моделирования, которые можно использовать для просмотра содержимого файла анимации.

3. Технические характеристики

3.1. Постановка задачи на разработку программы

Цель работы - реализовать систему неявной скелетной анимации.

Основные задачи работы:

1. Создание исходных данных (файлов) для неявной скелетной анимации.
2. Загрузка анимации из файла (содержание описанно в ТЗ).
3. Рассчет кадров анимации.
4. Воспроизведение анимации на экране средствами OpenGL.

Второстепенные задачи работы:

1. Предоставление пользователю возможности перейти к любому моменту времени в анимации.
2. Отрисовка костей модели.
3. Возможность вкл./выкл. учет нормалей и материалов во время отрисовки.
4. Поддержка двух видов камер в OpenGL. Первый вид это камера, движение которой сковано орбитой вокруг модели, и другой тип это камера двигающаяся совершенно свободно.

3.2. Описание алгоритма и функционирования программы

3.2.1. Выбор алгоритма

3.2.1.1. Различные подходы

к созданию систем 3-х мерной анимации балансируют между методами с большим количеством вычислений и методами, требующими большого объема памяти. Условно можно выделить явные и неявные системы анимации.

3.2.1.2. Явные системы анимации

хранят отдельную модель для каждого кадра. После записи данных в файл, существует много методов для воспроизведения анимации. Такие методы требуют лишь элементарной математики. Однако типичная запись одного трэка анимации для одного персонажа занимает около 10MB (в формате MD3).

Рис. 1: Каждому кадру соответствует своя модель

Предпочтение явным системам отдается, когда необходимо анимировать большие группы людей или животных.

3.2.1.3.Неявные системы анимации

хранят не модели, а более высокоуровневое описание движения. В частности **неявные системы скелетной анимации** содержат описание (через матрицу поворота) для каждой кости, как например локоть, плечо, шея. В реальном времени эти описания применяются к неанимированной модели для рассчета следующего кадра анимации. Эти рассчеты обычно требуют сложной математики с матрицами и тригонометрией. А следовательно и много CPU времени.

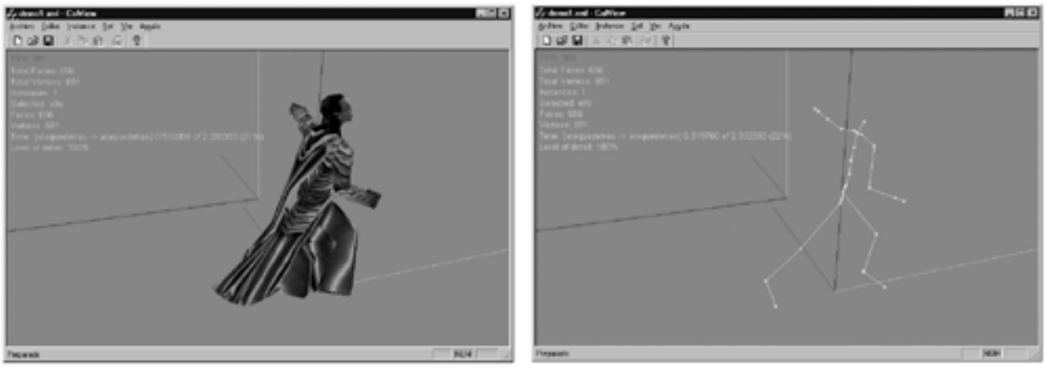

Рис. 2: Слева: анимированный персонаж; справа: скелет для данного кадра

Неявные системы используются, когда действия персонажа связанны с другими предметами и нельзя предугадать все возможные варианты анимации. Например для того, чтобы ставить ступню параллельно поверхности при движении по неровной земле.

3.2.2. Описание алгоритма и структур данных

Для реализации требуются 3 вещи:

Скелет Скелет определяет иерархию частей тела персонажа. Скелет, - это дерево из костей.

Трэк анимации В трэке содержатся матрицы поворота скелета в ключевые моменты времени. Трэк, - это массив пар (время, набор матриц поворота).

Модель - это полигональная модель.

Для создания эффекта анимации необходимо извлекать из трэка набор матриц поворота соответствующий настоящему моменту времени. Применять этот набор матриц к скелету, а затем применять позу скелета к модели.

3.2.2.1.Применение описания из трэка к скелету

Матрицы поворота для всех костей записываются в трэке относительно матрицы поворота родителя. Поэтому для деформации скелета необходимо применять матрицы последовательно. Мы начинаем с корневой кости и применяем к ней описанную в трэке анимации матрицу поворота. Затем, мы двигаемся вглубь скелета применяя матрицу родителя и матрицу описанную в трэке (то есть считаем глобальную матрицу поворота для данной кости), пока не рассмотрим все кости.

3.2.2.2.Применение деформации скелета к модели

После того как рассчитанны матрицы поворотов для скелета, их необходимо применить на вершины модели. Для этого используется рекурсивный алгоритм очень похожий на предыдущий (п. 3.2.2.1).

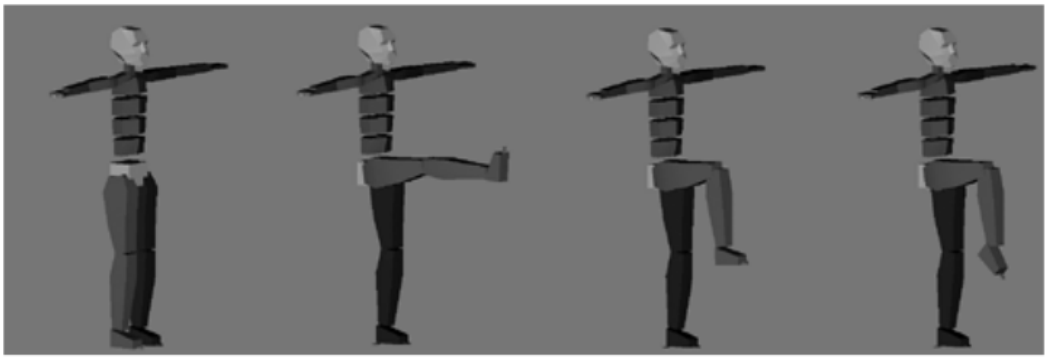

Рис. 3: Применение преобразований, начиная от копчика (корневой кости) и заканчивая ступней.

```
deform (bone root, mesh original, mesh deformed)
  for each child_bone of root
    for each vertex in the original mesh
      if bone_weight > 0
          apply bone global transform to vertex
          scale the resulting point by the bone weight
          store the result in deformed
      end if
    end for
    if child_bone has children
      deform (children of this node, mesh original, deformed)
    end if
  end for
```

3.2.3. Реализация системы скелетной анимации

Для реализации было созданно несколько функциональных блоков. Диаграмма основных блоков:

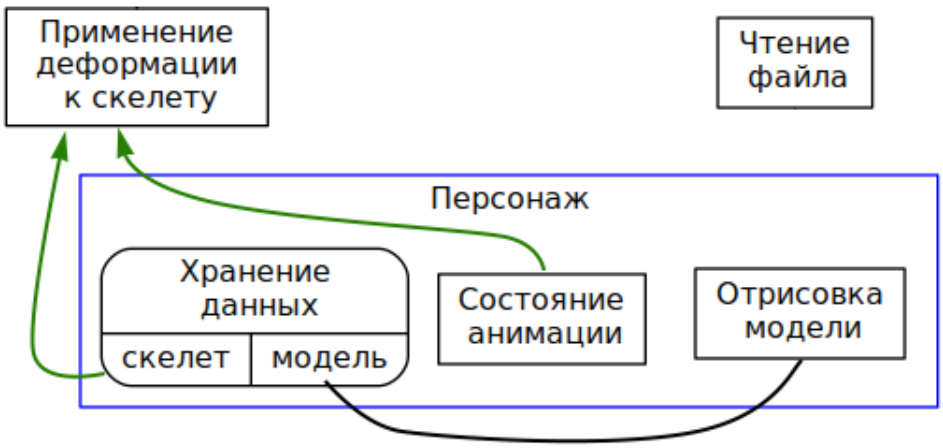

Рис. 4: Схема логических блоков в программе

3.2.3.1. Описание системы, блок чтения файла

Заполняет структуры данными из файла. С помощью библиотеки Assimp производится чтение из файла. Для оптимальной работы данные перераспределяются из структур Assimp в свои. Другие функции

```
public void LoadScene(byte[] filedata)
{
    using (MemoryStream fs = new MemoryStream(filedata))
    {
        _cur_scene = new SceneWrapper(ReadAssimpScene(fs, "dae"));
        // у входных данных всегда только один трэк анимации
        _animation_track = _cur_scene.Animations[0];
        // блок применения описаний из трэка анимации к скелету
        _action = new Animator(_animation_track);
        // корневая кость скелета
        BoneNode bones = _cur_scene.BuildBoneNodes("Armature");
        // модель
        Node mesh = _cur_scene.FindNode("Mesh");
        // блок состояния анимации
        ActionState state = new ActionState();
        // блок персонажа
        _enttity = new Entity(_cur_scene, mesh, bones, state);
    }
}
```

этой библиотеки не используются. Ниже приведен упрощенный код считывающий информацию из файла и строящий стуктуры данных.

3.2.3.2.Описание системы, блок состояния анимации

Хранит состояние анимации. Наиболее важные поля:

- Название трэка анимации.
- Настоящий момент времени в секундах.
- Индексы всех ключевых кадров и время каждого ключевого кадра.

Есть функция SetTime(. . .) для перехода к определенному моменту времени. Она находит интервал между ключевыми кадрами, подсчитывает величину интерполяции.

```
// функция в блоке состояния анимации для прыжка к определенному времени
public void SetTime(double time_seconds)
{
    double time_ticks = time_seconds * TickPerSec;
    // когда время в секундах переполняеться, запускаем анимацию с начала
    double time = time_ticks % TotalDurationTicks;
    // поиск интервала между ключевыми кадрами
    int start_frame = FindStartFrameAtTime(time_seconds);
    int end_frame = (start_frame + 1) % KeyframeCount;
    // нахождение значения для интерполяции между ключевыми кадрами
    double delta_ticks = KeyframeTimes[end_frame] - KeyframeTimes[start_frame];
    // если начали анимацию заново
    if (delta_ticks < 0.0)
    {
        delta_ticks += TotalDurationTicks;
    }
    double blend = (time - KeyframeTimes[start_frame]) / delta_ticks;
```

```
    // приписываем результаты рассчетов
    OriginKeyframe = start_frame;
    TargetKeyframe = end_frame;
    KfBlend = blend;
}
```

3.2.3.3.Описание системы, блок хранения данных

Работает со скелетом и моделью. Реализует функции поиска костей в скелете или подмоделей в модели. Функция BuildBones строит скелет по данным из модели (скелет как отдельный класс не существует, он определяеться корневой костью).

Ниже приведен класс описывающий кость скелета:

```
class BoneNode
{
    public string Name;
    public Matrix4 GlobalTransform;
    public Matrix4 LocalTransform;

    public BoneNode Parent;
    public List<BoneNode> Children;
    public BoneNode(Node assimp_node) { ... }
}
```

3.2.3.4.Описание системы, блок деформации скелета

Применяет данные, описывающие (в матрицах поворота) новую позицию для каждой кости к костям из скелета. То есть деформирует скелет в соответствии с моментом времени в анимации. На вход блока подается класс ActionState, содержащий информацию о состоянии анимации и корневая кость скелета.

```
// функция для извлечения матриц поворота из трэка и применения их к скелету
public void ChangeLocalFixedDataBlend(ActionState st)
{
    // для каждой кости создает свой канал анимации
    foreach (NodeAnimationChannel channel in _action.NodeAnimationChannels)
    {
        BoneNode bone_nd = _scene.GetBoneNode(channel.NodeName);
        // поворот кости
        Quaternion target_roto = Quaternion.Identity;
        if (channel.RotationKeyCount > st.TargetKeyframe)
        {
            target_roto = channel.RotationKeys[st.TargetKeyframe].Value.eToOpenTK();
        }
        Quaternion start_frame_roto = channel.RotationKeys[st.OriginKeyframe].Value;
        // интерполяция поворота между двумя ключевыми кадрами
        Quaternion result_roto = Quaternion.Slerp(start_frame_roto, target_roto, (float)st.KfBlend);
        // сдвиг кости
        Vector3 target_trans = Vector3.Zero;
        if (channel.PositionKeyCount > st.TargetKeyframe)
        {
            target_trans = channel.PositionKeys[st.TargetKeyframe].Value;
        }
```

```
        Vector3 cur_trans = channel.PositionKeys[st.OriginKeyframe].Value;
        // интерполяция сдвига между двумя ключевыми кадрами
        Vector3 result_trans = cur_trans + Vector3.Multiply(target_trans - cur_trans, (float)st.KfBlend);
        // объединение поворота и сдвига
        Matrix4 result = Matrix4.CreateFromQuaternion(result_roto);
        result.Row3.Xyz = result_trans;
        bone_nd.LocalTransform = result;
    }
}

// функция для рассчета глобальной матрицы поворота для каждой кости
// эта матрица будет позднее применена к вершинам модели
private void ReCalculateGlobalTransform(BoneNode nd)
{
    nd.GlobalTransform = nd.LocalTransform * nd.Parent.GlobalTransform;
    foreach (var child in nd.Children)
    {
        ReCalculateGlobalTransform(child);
    }
}
```

3.2.3.5.Описание системы, блоки отрисовки модели

Загружает данные о модели в OpenGL. Запрашивает OpenGL об отводе буферов памяти под вершины, нормали, цвета вершин и массив индексов. Применяет свойства материала, например: цвет, коэффициент рассеивания света, коэффициент свечения и т.д.

Данные о вершинах, материалах и нормалях необходимо загружать в буферы памяти расположенные на видеокарте для того что бы обеспечить приложению приемлимую скорость отрисовки. Ниже приведен код для загрузки данных в память видеокарты:

```
// объект содержащий номера буферов в OpenGL
struct Vbo
{
    public int VertexBufferId;
    public int NormalBufferId;
    public int ElementBufferId;
    public int NumIndices;
}

// функция для создания нового буфера
// и заполнения его данными из массива векторов
private void NewOpenGLBufferWithFloats(out int outGlBufferId, List<Vector3D> dataBuffer)
{
    GL.GenBuffers(1, out outGlBufferId);
    GL.BindBuffer(BufferTarget.ArrayBuffer, outGlBufferId);
    int sizeof_vec3d = 12; // X,Y,Z = 3 floats, 4 bytes each
    var byteCount = dataBuffer.Count * sizeof_vec3d;
    var temp = new float[byteCount];
    var n = 0;
    foreach(var v in dataBuffer)
    {
        temp[n++] = v.X;
        temp[n++] = v.Y;
```

```
        temp[n++] = v.Z;
    }
    GL.BufferData(BufferTarget.ArrayBuffer, (IntPtr)byteCount, temp, BufferUsageHint.StreamDraw);
    VerifyArrayBufferSize(byteCount);
    GL.BindBuffer(BufferTarget.ArrayBuffer, 0);
}

// функция для загрузки данных о модели в память видеокарты
private void Upload(out Vbo vboToFill)
{
    vboToFill = new Vbo();
    NewOpenGLBufferWithFloats(out vboToFill.VertexBufferId, _mesh.Vertices);
    if (_mesh.HasNormals)
    {
        NewOpenGLBufferWithFloats(out vboToFill.NormalBufferId, _mesh.Normals);
    }
}
```

Для создания эффекта движения необходимо каждый кадр менять содержимое буферов расположыенных на видеокарте. А именно необходимо менять координаты вершин и направления нормалей к каждой вершине (для корректного отображения света/тени). Для этого необходимо послать запрос к драйверу OpenGL и получить указатель на память с загруженными ранее данными. Далее приведен код модифицирущий данные в буфере для следующего кадра.

```
// функция в блоке отрисовки модели для получения доступа к буферу OpenGL
public void BeginModifyNormalData(out IntPtr data, out int qty_normals)
{
    GL.BindBuffer(BufferTarget.ArrayBuffer, _vbo.NormalBufferId);
    data = GL.MapBuffer(BufferTarget.ArrayBuffer, BufferAccess.ReadWrite);
    qty_normals = _mesh.Normals.Count;
}
// функция в блоке отрисовки модели для освобождения буфера OpenGL
public void EndModifyNormalData()
{
    bool data_upload_ok = GL.UnmapBuffer(BufferTarget.ArrayBuffer);
    if (! data_upload_ok)
    {
        throw new Exception("OpenGL driver has failed.");
    }
}

// функция в блоке персонажа для применения матриц костей к вершинам модели
public void RecursiveTransformVertices(Node node)
{
    foreach (int mesh in nd.Meshes)
    {
        // получаем указатель на буфер в OpenGL
        IntPtr pbuf_opengl;
        int qty_vertices;
        mesh.BeginModifyVertexData(out pbuf_opengl, out qty_vertices);
        // изначальная модель без деформаций
        Mesh original_data = mesh.OriginalData;
        // go over every vertex in the mesh
        unsafe
```

```
        {
            int sz = 3;          // размер шага
            float* coords = (float*)pbuf_opengl;
            for (int vertex_id = 0; vertex_id < qty_vertices; vertex_id++)
            {
                Matrix4 matrix_with_offset = mesh._vertex_id2matrix[vertex_id];
                // получить изначальные координаты вершины
                Vector3 vertex_default = original_data.Vertices[vertex_id];
                Vector3 vertex;
                Vector3.Transform(ref vertex_default, ref matrix_with_offset, out vertex);
                // Применение веса к вершине
                Vector3 delta = vertex_default - vertex;
                vertex += delta *  mesh._vertex_id2bone_weight[vertex_id];
                // Запись новых координат обратно в буфер OpenGL
                coords[vertex_id*sz + 0] = vertex.X;
                coords[vertex_id*sz + 1] = vertex.Y;
                coords[vertex_id*sz + 2] = vertex.Z;
            }
        }
        mesh.EndModifyVertexData();

        foreach (Node child in nd.Children)
        {
            RecursiveTransformVertices(child);
        }
    }
}
```

3.2.3.6.Описание системы, блок прерсонажа

Объединяет компоненты необходимые для анимации одного персонажа. Хранит ссылки на скелет (корневую кость), состояние анимации (ActionState), на саму модель и на класс отрисовки модели (MeshDraw). В частности блок персонажа применяет трансформации из скелета к вершинам модели (взвешивая действие каждой кости на вершину) и модифицирует данные в буфере данных OpenGL, что и создает эффект анимации.

3.3. Метод организации входных и выходных данных

3.3.1.Описание метода входных и выходных данных

Входными данными является файл в формате collada (.dae) , в котором в обязательном порядке должны присутствовать следующие элементы:

1. Одна полигональная модель.
2. Один трэк анимации.
3. Один скелет.

Если не выполненны условия на наличие полигональной модели, трэка анимации и связанного с моделью скелета то у программы не хватит информации для воспроизведения анимации.

Выходными данными является отображение анимации на экране.

3.4. Выбор состава технических средств

3.4.1. Состав технических и программных средств

Для возможности запустить приложение необходимо учесть следующие системные требования:

1. Компьютер, оснащенный:
   (a) Обязательно 64-разрядный (x64) процессор с тактовой частотой 1 гигагерц (ГГц) или выше;
   (b) 512 мегабайт (ГБ) оперативной памяти (ОЗУ);
   (c) 2 ГБ (для 64-разрядной системы) пространства на жестком диске;
   (d) графическое устройство OpenGL с драйвером версии 3.1 или выше.
2. Монитор
3. Видеокарта
4. Мышь
5. Клавиатура

Также необходимо учесть следующие програмные требования:

1. Поддержка OpenGL версии 3.1
2. 64-битная операционная система Windows 7.
3. .NET Framework версии 4.5.1
4. Библиотека Assimp версии 3.1
5. Библиотека OpenTK версии 1.1.4

Программа была протестирована и отлажена на версии OS Windows 7 с использованием .Net Framework 4.5.1, OpenTK версии 1.1.4 и Assimp версии 3.1.

Качество и корректность работы программы при других версиях библиотек и операционных систем не проверялось.

Программа использует буферы графической памяти типа STREAM_WRITE и функции glMapData и glSubBufferData которые в OpenGL официально поддерживаются лишь с версии 3.1

Технические требования к памяти и периферии не превышают технических требований к операционной системе Windows 7 с установленным на ней .Net Framework 4.5.1

4. Технико-экономические показатели

4.1. Ориентировачная экономическая эффективность и годовая потребность

В силу простоты интерфейса она подходит для использования людям не очень хорошо знакомым с более мощными и трудными в использовании пакетами для 3-х мерного моделирования, которые можно использовать для просмотра содержимого файла анимации. Предполагается, что программа будет использоваться пользователем несколько раз в неделю, на протяжении коротких периодов времени, т. е. количество сеансов на одном рабочем месте в год составит примерно 48 сеансов. Она может использоваться в процессе отладки приложений использующих анимацию и в работе дизайнера 3D моделей.

4.2. Экономические преимущества разработки

Экономические преимущества разработки в сравнении с лучшими отечественными и зарубежными аналогами рассчитаны на январь 2016 года. Существующими аналогами данного приложения являются пакеты для 3-х мерного моделлирования и анимации. В силу того что данное приложение распростроняется бесплатно, единственным экономически выгодным аналогом к нему будет программа Blender. Однако Blender гораздо более сложен в использовании и потребляет намного больше системных ресурсов (жесткой памяти, ОЗУ, времени процессора).

5. Источники, используемые при разработке

5.1. Список используемой литературы

1. OpenGL Superbible: Comprehensive Tutorial and Reference (7th Edition) Graham Sellers (Author), Richard S Wright Jr. (Author), Nicholas Haemel (Author) ISBN-13: 978-0672337475
2. Порев В.Н. Компьютерная графика. – СПб.: БХВ-Петербург, 2002. – 432 с.: ил.
3. ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению // Единая система программной документации. -М.:ИПК Издательство стандартов, 2001.
4. ГОСТ 19.101-77 Виды программ и программных документов //Единая система программной документации. -М.: ИПК Издательство стандартов, 2.: 001.
5. ГОСТ 19.103-77 Обозначения программ и программных документов. //Единая система программной документации. -М.: ИПК Издательство стандартов, 2001.
6. Требования к системе для .NET Framework 4.5. [Электронный ресурс]// URL: https://msdn.microsoft.com/ru-ru/library/8z6watww(v=vs.110).aspx (Дата обращения: 23.02.2016, режим доступа: свободный).
7. Системные требования ОС Windows 7. [Электронный ресурс]// URL: http://windows.microsoft.com/ru-ru/windows7/products/system-requirements (Дата обращения: 20.08.2016, режим доступа: свободный).
8. Документация OpenGL 3.3 [Электронный ресурс] // https://www.opengl.org/sdk/docs/man/ (Дата обращения: 21.10.2016, режим доступа: свободный)

6. Приложение 1. Терминология

6.1. Терминология

Корневая вершина (англ. root node) Самый верхний узел дерева.

Полигональная сетка (жарг. меш от англ. polygon mesh) Совокупность вершин, рёбер и граней, которые определяют форму многогранного объекта в трехмерной компьютерной графике и объёмном моделировании. Гранями являются треугольники.

Дерево Связный ациклический граф. Связность означает наличие путей между любой парой вершин, ацикличность — отсутствие циклов и то, что между парами вершин имеется только по одному пути.

Степень вершины Количество инцидентных ей (входящих/исходящих из нее) ребер.

Интерполяция, интерполирование анимации Способ нахождения промежуточных значений состояния анимации по имеющемуся дискретному набору известных значений.

Z-буферизация В компьютерной трёхмерной графике способ учёта удалённости элемента изображения. Представляет собой один из вариантов решения «проблемы видимости»

Z-конфликт (англ. Z–fighting) Если два объекта имеют близкую Z-координату, иногда, в зависимости от точки обзора, показывается то один, то другой, то оба полосатым узором.

OpenGL (Open Graphics Library) Спецификация, определяющая независимый от языка программирования платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику. На платформе Windows конкурирует с Direct3D.

Рендеринг (англ. rendering — «визуализация») Термин в компьютерной графике, обозначающий процесс получения изображения по модели с помощью компьютерной программы.

Текстура Растровое изображение, накладываемое на поверхность полигональной модели для придания ей цвета, окраски или иллюзии рельефа. Приблизительно использование текстур можно легко представить как рисунок на поверхности скульптурного изображения.

Лист регистрации изменений

Изм.	Номера листов (страниц)				Всего листов (страниц) в докум.	№ докум.	Входящий № сопроводительного докум. и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					