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

_____ Р. З. Ахметсафина

”____”_____ 2016 г

Утверждаю

Академический руководитель
образовательной программы
«Программная инженерия»

_____ В. В. Шилов

”____”_____ 2016 г

ПРОГРАММА СКЕЛЕТНАЯ АНИМАЦИЯ

Пояснительная записка

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.509000 ТЗ 01-1-ЛУ

Студент группы БПИ 151 НИУ ВШЭ
_____ Абрамов А.М.
”____”_____ 2016 г

2016

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

УТВЕРЖДЕНО
RU.17701729.509000 ТЗ 01-1-ЛУ

ПРОГРАММА СКЕЛЕТНАЯ АНИМАЦИЯ

Пояснительная записка

RU.17701729.509000 ТЗ 01-1-ЛУ

Листов 25

2016

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

Содержание

1. Введение

1.1. Наименование

Наименование: «Программа скелетная анимация».
Наименование на английском: «Program of Skeletal Animation».

1.2. Краткая характеристика

Цель работы - реализовать программу скелетной анимации. В задачи работы входит загрузка анимации из файлов collada (.dae), расчет промежуточных кадров анимации и воспроизведение анимации на экране средствами OpenGL. Также программа предоставляет пользователю возможность менять положение камеры, перейти к любому моменту времени в анимации и просмотреть иерархию костей и модели. В состав работ также входит создание демонстрационных исходных данных (файлов) для данной программы.

Файл анимации в формате collada, удовлетворяющий требованиям входных данных, может быт подготовлен пользователем в любом пакете для трех мерного моделирования. Например в программе Blender (https://www.blender.org/, разработчик: некоммерческая организация Blender Foundation)

1.3. Документы, на основании которых ведется разработка

Разработка программы ведется на основании приказа №6.18.1-02/1112-19 от 11.12.2015 «Об утверждении тем, руководителей курсовых работ студентов образовательной программы Программная инженерия факультета компьютерных наук» в соответствии с учебным планом подготовки бакалавров по направлению «Программная инженерия», факультета Компьютерных наук, Национального исследовательского университета «Высшая школа экономики»

2. Назначение разработки

2.1. Функциональное назначение

Функциональным назначением приложения является предоставление пользователю возможности загрузить 3D анимацию из файла collada (.dae), проиграть ее, отобразить анимацию учитывая связанные с ней материалы и нормали, просмотреть модели и кости входящие в состав анимации, перейти к любому моменту времени в анимационном файле, изменить положение камеры.

2.2. Эксплутационное назначение

Программа предназначена для запуска на персональном компьютере с операционной системой семейства Windows. Она может использоваться в учебных целях для демонстации основных компонентов систем скелетной анимации. Она может использоваться программистом в процессе отладки приложений использующих анимацию. Ею может воспользоваться любой человек, желающий просмотреть записанную в файле анимацию, но не знакомый со сложными интерфейсами пакетов для трех мерного моделирования.

3. Технические характеристики

3.1. Постановка задачи на разработку программы

Цель работы - реализовать программу скелетной анимации.

Задачи работы:

1. Создание исходных данных (файлов) для скелетной анимации.
2. Загрузка анимации из файла (содержание описанно в ТЗ).
3. Рассчет кадров анимации.
4. Воспроизведение анимации на экране средствами OpenGL.
5. Предоставление пользователю возможности перейти к любому моменту времени в анимации.
6. Отрисовка костей модели.
7. Возможность вкл./выкл. учет нормалей и материалов во время отрисовки.
8. 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 программ трех 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 трека 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 расчета следующего кадра анимации. Эти расчеты обычно требуют сложной математики с матрицами и тригонометрией. А следовательно и много CPU времени.

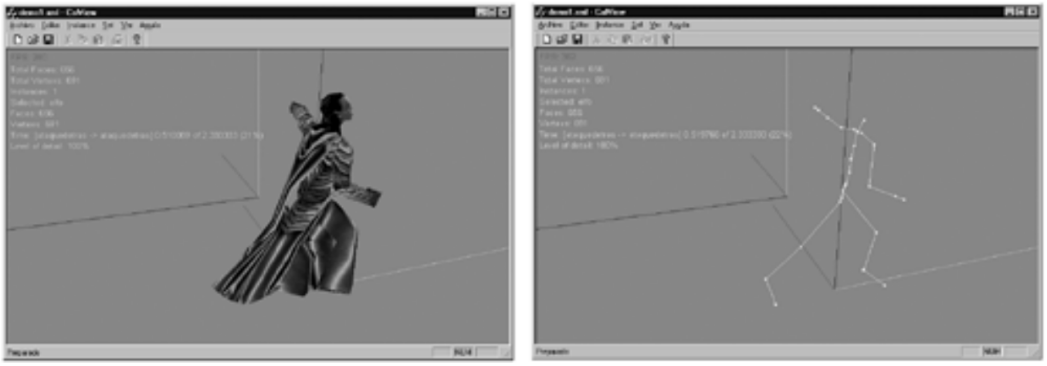

Рис. 2: Слева: анимированный персонаж; справа: скелет для данного кадра

Неявные системы используются, когда персонаж может совершать несколько действий одновременно и невозможно предугадать все возможные варианты анимации.

3.2.2. Основные определения и структуры данных

3.2.2.1. Кость

Каждая кость содержит информацию о трех мерной трансформации (которая состоит из поворота, растяжения и смещения), а также информацию о кости-отце. Глобальная трансформация кости-потомка, - это произведение глобальной трансформации кости-родителя на локальную трансформацию самой кости-потомка. Изменение транформации родителя, меняет трансформацию потомка.

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 node_data) { ... }
}
```

3.2.2.2. Скелет

Скелетом называют иерархичную (деревообразную) структуру сформированную костями. Скелет определяеться с помощью корневой кости в иерархии.

3.2.2.3. Трек анимации

В треке содержатся матрицы поворота скелета в ключевые моменты времени. В упрощенном виде трек можно представить в качестве массива пар: { время ключевого кадра, массив из матриц поворота для каждой кости }

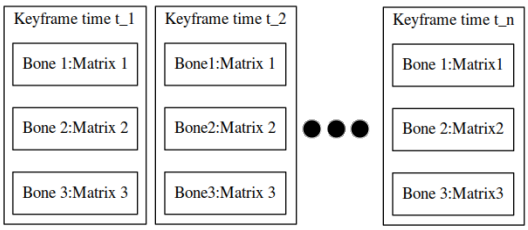

Рис. 3: В ключевые моменты времени (t_1, t_2, ... t_n), каждая кость ставиться в соответствие с матрицей локальной трансформации.

3.2.2.4. Модель

Модель состоит из набора вершин, весов вершин, нормалей, материалов. В пакете для трех мерной анимации модель привязывают к скелету, каждая вершина модели «привязывается» к какой-либо кости скелета (или к нескольким костям). После привязки модели к скелету при движении отдельной кости будут двигаются и все вершины, привязанные к ней.

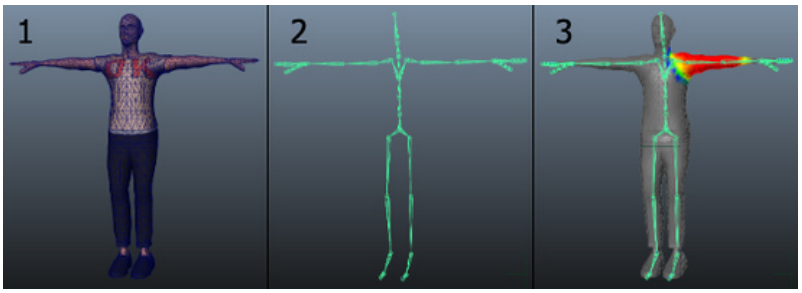

Рис. 4: Слева: модель; в центре: скелет; справа: выделенны вершины модели, которые были привязанны к кости правого предплечья

3.2.3. Описание алгоритма

Для создания эффекта анимации необходимо извлекать из трека набор матриц поворота соответствующий настоящему моменту времени. Применять этот набор матриц к скелету, а затем применять позу скелета к модели (то есть изменять координаты вершин).

3.2.3.1.Применение данных из трека к скелету

Матрицы поворота для всех костей записываются в треке относительно матрицы поворота родителя. Поэтому для анимации скелета необходимо применять матрицы в последовательном порядке. Начинаем с корневой кости и применяем к ней описанную в треке анимации матрицу поворота. Затем, двигаемся вглубь скелета по иерархии и находим произведение матрицы родителя и матрицы потомка (извлеченной из трека). Другими словами расчитывается глобальная матрица поворота для кости-потомка.

Необходимо продолжать движение по иерархии пока не будут рассмотренны все кости. Ниже приведен псевдокод для применения данных из трека к скелету:

```
deform_skeleton (bone root, matrix global, track matrices)
  get the matrix for root bone from track matrices
  multiply it by matrix global
  store the result in bone root as global transform
  if root has children
    deform (children of this node, root bone global matrix, track matrices)
  end if
end function
```

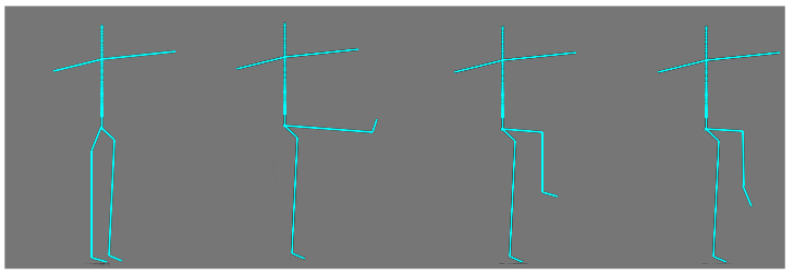

Рис. 5: Последовательное применение преобразований, начиная от копчика (корневой кости) и заканчивая ступней.

3.2.3.2.Применение скелета к модели

После того как рассчитанны матрицы поворотов для скелета, их необходимо применить на вершины модели. Для этого используется рекурсивный алгоритм похожий на алгоритм для анимации скелета. Ниже приведен его псевдокод:

```
deform_mesh (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_mesh (children of this node, mesh original, deformed)
    end if
  end for
end function
```

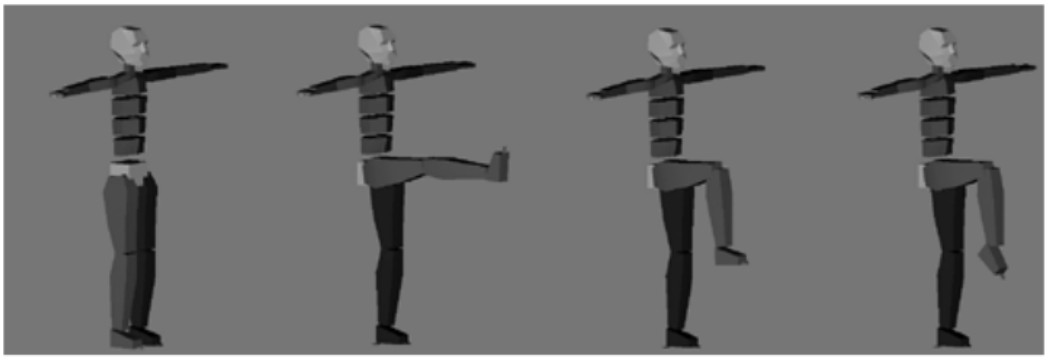

Рис. 6: Применение преобразований к вершинам модели.

3.2.4. Реализация программы скелетной анимации

В программе можно выделить несколько логических блоков. Каждый блок состоит из одного или более классов предоставлающих определенный функционал.

1. Блок чтения данных.
2. Блок хранения состояния анимации.
3. Блок хранения данных модели и скелета.
4. Блок деформации скелета.
5. Блок отрисовки модели.
6. Блок управления компонентами.

3.2.4.1. Чтение данных

С помощью библиотеки Assimp производится чтение из файла. Для оптимальной работы данные перераспределяются из структур Assimp в свои. Другие функции этой библиотеки не используются. Ниже приведен упрощенный код считывающий информацию из файла и строящий стуктуры данных.

```
// функция для загрузки данных из библиотеки Assimp, в созданные структуры данных
public void LoadScene(byte[] filedata)
{
    using (MemoryStream fs = new MemoryStream(filedata))
    {
        _cur_scene = new SceneWrapper(ReadAssimpScene(fs, "dae"));
        // во входных данных всегда только один трек анимации
        _animation_track = _cur_scene.Animations[0];
        // примененает описания из трека анимации к скелету
        _action = new Animator(_animation_track);
        // корневая кость скелета
        BoneNode bones = _cur_scene.BuildBoneNodes("Armature");
        // модель
        Node mesh = _cur_scene.FindNode("Mesh");
```

```
        // хранит состояние анимации
        ActionState state = new ActionState();
        // объединет компоненты
        _enttity = new Entity(_cur_scene, mesh, bones, state);
    }
}
```

3.2.4.2.Хранение состояния анимации

Класс ActionState хранит состояние анимации. Наиболее важные поля:

- Трек анимации.
- Настоящий момент времени в секундах.
- Массив времени для всех ключевых кадров.

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

```
// функция 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 необходимо запустить анимацию заново
    if (delta_ticks < 0.0)
    {
        delta_ticks += TotalDurationTicks;
    }
    double blend = (time - KeyframeTimes[start_frame]) / delta_ticks;
    // приписываем результаты расчетов
    OriginKeyframe = start_frame;
    TargetKeyframe = end_frame;
    KfBlend = blend;
}
```

3.2.4.3.Хранение данных модели и скелета

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) { ... }
}
```

3.2.4.4.Деформация 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 трека 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 расчета 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);
    }
}
```

3.2.4.5.Отрисовка 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 идентификационные 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 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 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.");
    }
}
```

3.2.4.6.Управление компонентами

Класс Entity 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.

```
// функция в блоке управления компонентами 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);
        }
```

```
    }
}
```

3.3. 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 трек анимации.
3. Один скелет.

Если не выполненны условия на наличие полигональной модели, трека 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 програмных 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 программа 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 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.

7. Приложение 2. Формат Collada (.dae)

7.1. Формат Collada (.dae)

Collada — это формат, разработанный для обмена информацией между 3D приложениями. Управляется некоммерческой организацией Khronos Group. Collada использует открытый стандарт XML для обмена форматами, которые в противном случае были бы несовместимы. Collada был задуман как промежуточный формат для переноса файлов. Реализована поддержка в таких программах, как Maya, 3ds Max, Blender. Ниже приведен пример описания квадрата в формате collada:

```
<?xml version="1.0" encoding="utf-8"?>
<COLLADA xmlns="http://www.collada.org/2005/11/COLLADASchema" version="1.4.1">
  <asset>
    <unit name="meter" meter="1"/>
    <up_axis>X_UP</up_axis>
  </asset>
  <library_images/>
  <library_geometries>
    <geometry id="Plane_001-mesh" name="Plane.001">
      <mesh>
        <source id="Plane_001-mesh-positions">
          <float_array id="Plane_001-mesh-positions-array" count="12">-1 -1 0 1 -1 0 -1 1 0 1 1 0</float_array>
          <technique_common>
            <accessor source="#Plane_001-mesh-positions-array" count="4" stride="3">
              <param name="X" type="float"/>
              <param name="Y" type="float"/>
              <param name="Z" type="float"/>
            </accessor>
          </technique_common>
        </source>
        <source id="Plane_001-mesh-normals">
          <float_array id="Plane_001-mesh-normals-array" count="3">0 0 1</float_array>
          <technique_common>
            <accessor source="#Plane_001-mesh-normals-array" count="1" stride="3">
              <param name="X" type="float"/>
              <param name="Y" type="float"/>
              <param name="Z" type="float"/>
            </accessor>
          </technique_common>
        </source>
        <polylist count="2">
          <input semantic="VERTEX" source="#Plane_001-mesh-vertices" offset="0"/>
          <input semantic="NORMAL" source="#Plane_001-mesh-normals" offset="1"/>
          <vcount>3 3 </vcount>
          <p>1 0 3 0 2 0 0 0 1 0 2 0</p>
        </polylist>
      </mesh>
    </geometry>
  </library_geometries>
  <library_visual_scenes>
    <visual_scene id="Scene" name="Scene">
      <node id="Plane_001" name="Plane_001" type="NODE">
        <matrix sid="transform">69.99999 0 0 0 0 5.28485e-6 -69.99999 0 0 69.99999 5.28485e-6 0 0 0 0 1</matrix>
        <instance_geometry url="#Plane_001-mesh" name="Plane_001"/>
      </node>
    </visual_scene>
  </library_visual_scenes>
  <scene>
    <instance_visual_scene url="#Scene"/>
  </scene>
</COLLADA>
```

8. Приложение 3. Описание классов

8.1. Класс ActionState

Базовые классы:BaseForEventDriven.

8.1.1. Подробное описание

This class knows what argumets to pass to NodeInterpolator

8.2. Класс CameraDevice

Открытые члены

- Vector3 **ConvertScreen2WorldCoordinates** (Point screen_coords)
- Matrix4 **MatrixToOpenGL** ()

8.2.1. Подробное описание

Maintains camera abstraction.

8.3. Класс DrawConfig

Открытые атрибуты

- bool **EnableTexture2D** = false

8.3.1. Подробное описание

This class will be passed to the **Entity** (стр. 21)'s GetSettings() so it make the scene look best.

8.3.2. Данные класса

8.3.2.1. bool EnableTexture2D = false

OpenGL settings here is a template: public bool _enable = false;

8.4. Класс Entity

Открытые члены

- void **RecursiveCalculateVertexTransform** (Node nd, Matrix4x4 current)
- void **RenderModel** (**DrawConfig** settings)

8.4.1. Подробное описание

Represents the currently loaded object. We will have lots of these.

8.4.2. Методы

8.4.2.1. void RecursiveCalculateVertexTransform (Node *nd,* Matrix4x4 *current*)

bind tells the original delta in global coord, so we can find current delta

8.5. Класс Geometry

Открытые члены

- **Geometry** (IList< Mesh > scene_meshes, Node nd, BoneNode armature)

8.5.1. Подробное описание

Stores info on extra geometry of the entity.

8.6. Класс MeshDraw

Открытые члены

- **MeshDraw** (Mesh mesh, IList< Material > materials)
- void **RenderVBO** ()

8.6.1. Подробное описание

Mesh rendering using VBOs.

Based on http://www.opentk.com/files/T08_VBO.cs

8.6.2. Конструктор(ы)

8.6.2.1. **MeshDraw** (Mesh *mesh,* IList< Material > *materials*)

Uploads the data to the GPU.

8.6.3. Методы

8.6.3.1. void RenderVBO ( )

Render mesh from GPU memory. The pipeline is restored afterwards.

8.7. Класс MouseState

Открытые члены

- void **RecordMouseClick** (MouseEventArgs e)
- void **RecordMouseMove** (MouseEventArgs e)

Открытые атрибуты

- Point **ClickPos**
- Point **CurrentPos**
- readonly int **HorizHysteresis** = 4
- OpenTK.Vector3 **InnerWorldClickPos**
- OpenTK.Vector3 **InnerWorldPos**

8.7.1. Подробное описание

Simple class to store mouse status data. Monitor mouse status (delta, position, click_position, etc.)

8.8. Класс Renderer

Открытые члены

- void **DrawAxis3D** ()

8.8.1. Подробное описание

Class to control openGL settings and do the actual drawing. All openGL calls will be here.

8.8.2. Методы

8.8.2.1. void DrawAxis3D ( )

Important points to remember: Set normals. Must be clock wise vertex draw order The x-axis is accross the screen, so the Z-axis triangle must have component along X: +-1 since look at looks towards the center, we need to offset it a bit to see the Z axis.

8.9. Интерфейс ITransformState

Производные классы:CameraDrawing2D и CameraFreeFly3D.

8.9.1. Подробное описание

Implement this when class allows local matrix transforms. (**Entity** (стр. 21), Camera)

Лист регистрации изменений

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