ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕКИЙ УНИВЕРСИТЕТ
КАФЕДРА КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ И ДИЗАЙНА

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по курсу «Архитектура и проектирование графических систем»

на тему «Разработка графического редактора для работы с параметризованными трехмерными объектами»

Руководители:

_____ Карабчевский В.В.

«____»_____ 2020г.

_____ Доценко Г.В.

«____»_____2020г.

_____ Боднар А.В.

«____»_____2020г.

Выполнил:

студент группы ПИ-17в

_____ Петренко Д.А.

«____»_____2020г.

Донецк – 2020

РЕФЕРАТ

Курсовая работа содержит: 69 страниц, 21 рисунок

Целью курсового проектирования является создание приложения, представляющего из себя графический редактор для рендеринга трехмерных объектов и работы с ними, а так же закрепление и расширение теоретических знаний и практических навыков программирования, который должен показать способность и умение применять теоретические положения дисциплины «Архитектура проектирование графических систем», грамотно, самостоятельно и творчески решать задачи, четко и логично излагать свои мысли и решения, анализировать полученные результаты и делать необходимые выводы.

Задачей курсового проектирования является самостоятельное выполнение проектирования и разработки программного продукта в соответствии с техническим заданием. При этом должен показать свой уровень подготовки, умение выбрать и обосновать решение стоящих перед ним проблем, навыки работы с технической и справочной литературой, умение применять вычислительную технику в своей деятельности

Задачей проектирования является изучения основных аспектов работы в трехмерном пространстве и приминения полученных знаний на практике.

В результате выполнения курсовой работы был спроектирован и разработан графический редактор для работы с трехмерными объектами: конус, тороид, икосфера, куб, цилиндр. Была реализована возможность параметризации объектов, работы с камерой и сценой, выделение объекта курсором мыши, добавление и удаление объектов, смена режимов отрисовки.

ГРАФИЧЕСКИЙ РЕДАКТОР, ОБЪЕКТ, МАТРИЦА, ВЕКТОР, РАСТЕРИЗАЦИЯ, ПАРАМЕТРИЗАЦИЯ, ТРИАНГУЛЯЦИЯ

Содержание

ВВЕДЕНИЕ

Трёхмерная графика — раздел компьютерной графики, посвящённый методам создания изображений или видео путём моделирования объёмных объектов в трёхмерном пространстве.

3D-моделирование — это процесс создания трёхмерной модели объекта. Задача 3D-моделирования — разработать визуальный объёмный образ желаемого объекта.

Графическое изображение трёхмерных объектов отличается тем, что включает построение геометрической проекции трёхмерной модели сцены на плоскость с помощью специализированных программ.

Данная отрасль компьютерной графики на данный момент очень актуальна, так как применяется в большинстве сфер IT-технологий, связанных с компьютерной визуализацией.

Целью работы является разработка графического редактора для работы с трехмерными параметризованными объектами, заданного типа, позволяющего осуществлять следующие операции: поворот, перенос, масштабирование объекта; панорамирование, зуммирование; сохранение сцены в читабельную базу данных, работу с несколькими объектами, работу с камерой.

Для создания проекта была выбрана среда Microsoft Visual Studio и язык программирования C#.

1 РАЗРАБОТКА ПОЛИГОНАЛЬНОЙ МОДЕЛИ ОБЪЕКТА

1.1 Составляющие элементы объекта

Полигональное моделирование (polygonal modeling) — это самая первая разновидность трёхмерного моделирования, которая появилась в те времена, когда для определения точек в трёхмерном пространстве приходилось вводить вручную с клавиатуры координаты X, Y и Z. Как известно, если три или более точек координат заданы в качестве вершин и соединены рёбрами, то они формируют многоугольник (полигон), который может иметь цвет и текстуру. Соединение группы таких полигонов позволяет смоделировать практически любой объект. Недостаток полигонального моделирования состоит в том, что все объекты должны состоять из крошечных плоских поверхностей, а полигоны должны иметь очень малый размер, иначе края объекта будут иметь огранённый вид. Это означает, что если для объекта на сцене предполагается увеличение, его необходимо моделировать с большим количеством полигонов (плотностью) даже, несмотря на то, что большинство из них будут лишними при удалении от объекта.

В работе разрабатывается полигональная модель объекта «тренажер». Трехмерное изображение объекта представлено на рис. 1.1.

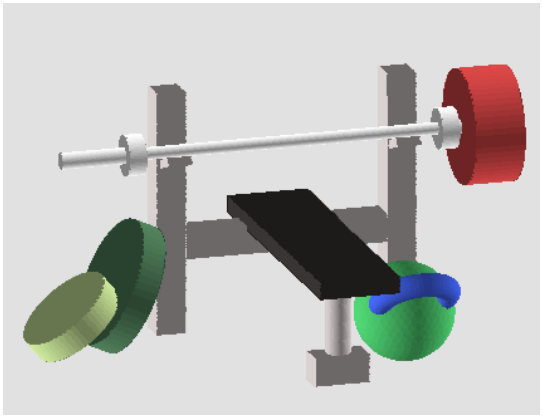


Рисунок 1.1 – Трехмерное изображение проектируемого объекта

Объект состоит из нескольких видов поверхностей, которые в свою очередь разбиваются на полигоны для удобства хранения и обработки.

Объект состоит из следующих видов поверхностей:

- параллелепипед;
- цилиндр;
- конус;
- тороид;
- икосфера.

Параллелепипед задается базовой точкой, шириной, длиной и высотой, на основе которых происходит триангуляция данной фигуры. Цилиндрические поверхности задаются точкой центра основания, радиусом и высотой. Количество полигонов в цилинде зависит от заданной точности его прорисовки. Полусфера задается точкой цента и радиусом. Триангуляция сферы происходит с помощью сферических координат, количество полигонов зависит от заданной точности прорисовки. Тороид задаётся внешним и внутренним радиусами, центральной точкой и радиусом кольца. Конус задаётся центром, длиной и шириной. Точность прорисовки выбрана оптимальной для достижения максимальной производительности.

1.2 Триангуляция поверхности объекта

Триангуляция поверхностей – это процесс разбиения сложных объектов на треугольные полигоны. Триангуляция удобна при программировании графики, т.к.:

- треугольник является простейшим полигоном, вершины которого однозначно задают грань;

- любую область можно гарантированно разбить на треугольники;

- вычислительная сложность алгоритмов разбиения на треугольники существенно меньше, чем при использовании других полигонов;

- реализация процедур визуализации более проста для области, ограниченной треугольником;

- для треугольника легко определить три его ближайших соседа, имеющих с ним общие грани.

В данном курсовом проекте над объектом производится триангуляция следующим образом.

Ящики разбиваются на треугольные грани, которые формируются между его вершинами.

Для триангуляции цилиндра, его основание в зависимости от заданной точности отрисовки, разбивается на определенное количество сегментов с помощью параметрического уравнения окружности. Триангуляция боковой поверхности аналогична триагуляции стороны ящика, триангуляция конуса аналогична.

Для триангуляции сферы используется формула параметрического уравнения сферы. При ее построении осуществляется проход по вершинам в оба направления, триангуляция сферы аналогична. Основными источниками ошибок угловых измерений в триангуляции являются инструментальные, личные и внешняя среда.

На множестве точек на плоскости задана триангуляция, если некоторые пары точек соединены ребром, любая конечная грань в получившемся графе образует треугольник, ребра не пересекаются, и граф максимален по количеству ребер.

2 ОПИСАНИЕ ВЫБРАННЫХ МЕТОДОВ И АЛГОРИТМОВ ВИЗУАЛИЗАЦИИ

2.1 Алгоритмы аффинных преобразований

Преобразование плоскости называется аффинным, если оно непрерывно, взаимно однозначно и таким образом любой прямой является прямая. Частными случаями аффинных преобразований являются движение и масштабирование объекта. Эти виды преобразований реализованы в курсовой работе. Афинные преобразования реализуются следующими шагами:

1) Сформировать матрицу перемещения;
2) Сформировать матрицу масштабирования и умножить на матрицу перемещения;
3) Сформировать матрицу поворота;
4) Сформировать матрицу модели, перемножив матрицу поворота с матрицей, получившейся на втором шаге;
5) Каждая точка объекта преобразуется с помощью полученной матрицы.

2.2 Алгоритм удаления скрытых линий

Для реалистичности изображаемого объекта применяется алгоритм удаления скрытых линий. В данной работе был использован алгоритм Z-буфера. Основной его принцип – обновление буфера, хранящего координаты z каждого пикселя изображения и его цвета.

Шаги алгоритма:

1) Заполнить *z*-буфер минимальным значением *z*;

2) Преобразовать каждый многоугольник в растровую форму в произвольном порядке;

3) Для каждого *Пиксел*(*x,y*) в многоугольнике вычислить его глубину *z*(*x,y*);

4) Сравнить глубину *z*(*x,y*) со значением *Zбуфер*(*x,y*), хранящимся в *z*-буфере в этой же позиции;

5) Если *z*(*x,y*) > *Zбуфер* (*x,y*), то записать атрибут этого многоугольника (интенсивность, цвет и т. п.) в буфер кадра и заменить *Zбуфер*(*x,y*) на *z*(*x,y*). В противном случае никаких действий не производить.

Достоинством алгоритма является относительная простота реализации и отсутствие сортировки. Недостаток – необходимость большого объема памяти для хранения буфера.

2.3 Алгоритмы проекционных преобразований

Проекционный метод изображения предметов основан на их зрительном представлении. Если соединить все точки предмета прямыми линиями (проекционными лучами) с постоянной точкой О (центр проекции), в которой предполагается глаз наблюдателя, то на пересечении этих лучей с какой-либо плоскостью получается проекция всех точек предмета. Соединив эти точки прямыми линиями в том же порядке, как они соединены в предмете, получим на плоскости перспективное изображение предмета или центральную проекцию.

Если центр проекции бесконечно удалён от картинной плоскости, то говорят о параллельной проекции, а если при этом проекционные лучи падают перпендикулярно к плоскости — то об ортогональной проекции.

2.3.1 Алгоритм параллельного проецирования

Параллельное проецирование можно рассматривать как частный случай центрального проецирования.

Если центр проекций при центральном аппарате проецирования перенести в бесконечность, то проецирующие лучи можно считать параллельными. Отсюда аппарат параллельного проецирования состоит из плоскости проекций П и направления Р. При центральном проецировании проецирующие лучи выходят из одной точки, а при параллельном проецировании — параллельны между собой.

В зависимости от направления проецирующих лучей параллельное проецирование может быть косоугольным, когда проецирующие лучи наклонены к плоскости проекций, и прямоугольным (ортогональным), когда проецирующие лучи перпендикулярны к плоскости проекций.

В данной работе используется перпендекулярное параллельное проецирование. Для него используется единичная матрица 4x4.

2.3.2 Алгоритм центрального проецирования

Центральной проекцией произвольной точки (x,y,z) называется точка пересечения плоскости проекции и луча, соединяющего точку наблюдения (xV ,yV ,zV) с точкой (x,y,z).

Центральные проекции классифицируются в зависимости от количества точек схода, которыми они обладают. В данном курсовом проекте реализована перспективная проекция с одной точкой схода. Проекция задается точкой наблюдения, точкой схода (центра), расстояниями от точки

наблюдения до передней и задней проецирующей плоскости (фокусное расстояние).

Шаги алгоритма:

1) Задается точка центра проецирования и точка наблюдения, а так же два фокусных расстояния.

2) В зависимости от этих параметров формируется матрица проекции.

3) Каждая точка объекта преобразуется с помощью полученной матрицы.

4) Отсекаются объекты, расположенные ближе переднего и дальше дальнего фокуса – они невидимы наблюдателю.

2.4 Алгоритм произвольных видовых преобразований (камеры)

Случай произвольного видового преобразования (камеры) представляет собой произвольное расположение картинной плоскости по отношению к объекту. По сути, задача сводится к преобразованию координат.

Так как камера движется по сфере вокруг точки центра, были использованы преобразования координат точки зрения по уравнениям, представленным на рисунке 2.1. Переменные ρ, θ, φ – это задание точки наблюдения в сферических координатах.

$$\begin{cases} x = \rho Sin\varphi \\ y = \rho Cos\varphi Sin\theta \\ z = \rho Cos\varphi Cos\theta \end{cases}$$

Рисунок 2.1 – Ортогональные координаты точки зрения

Так же в произвольном видовом преобразовании было использовано перспективное проецирование.

3 РАЗРАБОТКА СТРУКТУР ДАННЫХ ДЛЯ ХРАНЕНИЯ И ОПИСАНИЯ ОБЪЕКТА

3.1 Описание структур данных

Для описания объекта был реализован класс Shape, который хранит в себе информацию обо всех параметрах объеков, их положении на сцене и масштабировании, матрицу модели, а также массив полигонов, необходимый для построения объекта.

Структура класса Shape описана на рисунке 3.1.

```
8   public class Shape
9   {
10      public string type;
11
12      public float dx = 0;
13      public float dy = 0;
14      public float dz = 0;
15
16      public float rotation_x = 0;
17      public float rotation_y = 0;
18      public float rotation_z = 0;
19
20      public float scale_x = 10;
21      public float scale_y = 10;
22      public float scale_z = 10;
23
24      public System.Drawing.Color main_clr;
25      public System.Drawing.Color select_clr = Render.color(204, 204, 255);
26
27      public List<MyTriangle> triangles;
28  }
29 }
30
```

Рисунок 3.1 – Структура класса Shape

Класс MyTriangle реализует полигон, задающийся тремя вершинами и цветом растеризации. Структура этого класса представлена на рисунке 3.2.

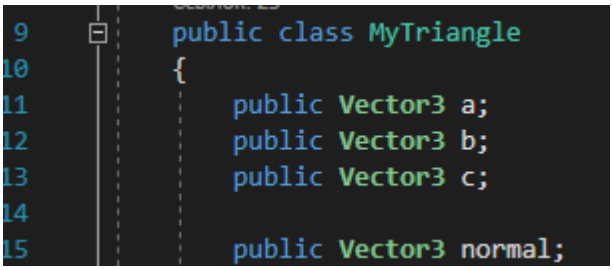


Рисунок 3.2 – Структура класса MyTriangle

3.2 UML-диаграммы классов

В разрабатываемой программной системе было выделено 6 основных классов:

MyMatrix – класс предназначенный для инициализации матриц размером 4x4 и реализации различных преобразований с ними.

Scene – класс, реализующий графический pipeline и хранение графичиских объектов.

Render - класс, реализующий стадию растеризации (функции рисования треугольника и линии).

MyTriangle – класс, реализующий полигон, задающийся тремя вершинами и цветом растеризации.

Shape – класс, предназначенный для описания объекта и реализации его параметрицазии.

GroupShapes – класс, реализующий группировку объектов в один

Диаграмма классов представлена на рисунке 3.3.

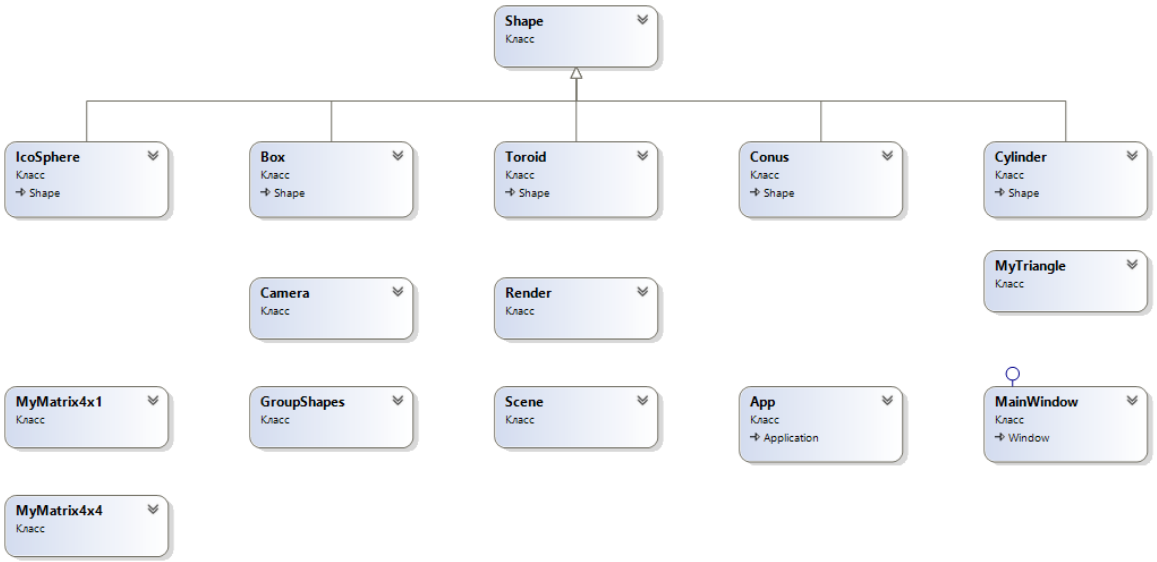


Рисунок 3.3 – Диаграмма отношений классов

4 РЕАЛИЗАЦИЯ ПРЕОБРАЗОВАНИЙ НАД ОБЪЕКТОМ

4.1 Реализация масштабирования

Масштабирование – это операция увеличения либо уменьшения параметров объекта по одному, двум или трем направлениям (осям). Операция относится к аффинным преобразованиям, реализовывается с помощью матриц преобразований. Матрица операции представлена на рисунке 4.1.

$$\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Рисунок 4.1 – Матрица операции масштабирования

В данном проекте масштабирование может производиться как по каждому направлению отдельно, так и по всем вместе. После формирования матрциы масштабирования в теле объекта Shape происходит пересчет матрицы модели, с помощью которой в последствие преобразуются все полигоны данного объекта.

Алгоритм масштабирования объекта на рисунке 4.2.

```
25      public static MyMatrix4x4 CreateScale(float x, float y, float z)
26      {
27          var temp = new MyMatrix4x4();
28          temp.points[0, 0] = x;
29          temp.points[1, 1] = y;
30          temp.points[2, 2] = z;
31          return temp;
32      }
```

Рисунок 4.2 – Алгоритм масштабирования

4.2 Реализация поворота

Поворот относится к аффинным преобразованиям. В трехмерной системе координат представляется возможным реализовать поворот объекта вокруг любой из осей координат. В данной работе реализован поворот вокруг каждой из координатных осей. Матрицы реализации поворота представлены на рисунке 4.3.

Поворот вокруг оси Z

$$\begin{bmatrix} Cos\theta & Sin\theta & 0 & 0 \\ -Sin\theta & Cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Поворот вокруг оси X

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & Cos\theta & Sin\theta & 0 \\ 0 & -Sin\theta & Cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Поворот вокруг оси Y

$$\begin{bmatrix} Cos\theta & 0 & -Sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ Sin\theta & 0 & Cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Рисунок 4.3 – Матрицы поворота

Матрица поворота учавствует в формировании матрицы модели. После каждого поворота объекта в классе Shape происходит пересчет его матрицы модели. Алгоритм поворота представлен на рисунке 4.4.

```
43          public static MyMatrix4x4 CreateRotationX(float A)
44          {
45              MyMatrix4x4 temp = new MyMatrix4x4();
46              temp.points[1, 1] = (float)Math.Cos(A);
47              temp.points[1, 2] = (float)Math.Sin(A);
48              temp.points[2, 2] = (float)Math.Cos(A);
49              temp.points[2, 1] = -(float)Math.Sin(A);
50              return temp;
51          }
52
            Ссылок: 2
53          public static MyMatrix4x4 CreateRotationY(float A)
54          {
55              MyMatrix4x4 temp = new MyMatrix4x4();
56              temp.points[0, 0] = (float)Math.Cos(A);
57              temp.points[2, 0] = -(float)Math.Sin(A);
58              temp.points[0, 2] = (float)Math.Sin(A);
59              temp.points[2, 2] = (float)Math.Cos(A);
60              return temp;
61          }
62
            Ссылок: 2
63          public static MyMatrix4x4 CreateRotationZ(float A)
64          {
65              MyMatrix4x4 temp = new MyMatrix4x4();
66              temp.points[0, 0] = (float)Math.Cos(A);
67              temp.points[1, 0] = -(float)Math.Sin(A);
68              temp.points[0, 1] = (float)Math.Sin(A);
69              temp.points[1, 1] = (float)Math.Cos(A);
70              return temp;
71          }
72
```

Рисунок 4.4 – Алгоритм поворота объекта

4.3 Реализация перемещения объекта

Перенос объекта является преобразованием движения. Он может выполняться как по любому из направлений координатных осей, так и в произвольном направлении. В данной работе реадизовано перемещение объекта в заданные координаты.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{bmatrix}$$

Рисунок 4.5 – Матрица переноса объекта

Матрица переноса является третьей составляющей матрицы модели, которая, аналогично предыдущим случаям аффинных преобразований, пересчитывается при каждом переносе. Алгоритм переноса представлен на рисунке 4.6.

```
34      public static MyMatrix4x4 CreateTranslation(float dx, float dy, float dz)
35      {
36          MyMatrix4x4 temp = new MyMatrix4x4();
37          temp.points[0, 3] = dx;
38          temp.points[1, 3] = dy;
39          temp.points[2, 3] = dz;
40          return temp;
41      }
42
```

Рисунок 4.6 – Алгоритм переноса объекта

4.4 Реализация изменения параметров

Параметризация объекта возможна как при его создании, так и в любое время работы программы. При создании параметризованного объекта, он создаётся в точке (0,0,0). Далее эти параметры сохраняются в структурах данных объекта, производится расчет зависимых характеристик. В любой момент пользователь может выделить объект мыкой и отредактировать его параметры. Форма для ввода параметров представлена на рис. 4.7.

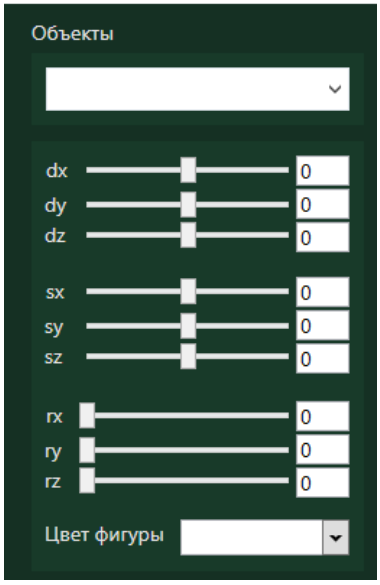


Рисунок 4.7 – Форма ввода параметров объекта

Параметризация уже созданного объекта проводится следующим образом:

1) Пользователь выбирает объект для параметризации;
2) Получает окно, отображающее текущие параметры объекта;
3) При необходимости изменяет желаемые параметры;
4) Объект перестраивается с новыми параметрами.

4.5 Реализация перспективной проекции

Перспективная проекция относится к центральным видам проекций, т.е. проекционные лучи направлены не параллельно друг другу, а сходятся в одной или нескольких точках. В данной работе перспективная проекция имеет одну точку схода. Матрица перспективной проекции формируется по алгоритму прдеставленному на рисунке 4.8. Параметры near и far представляют собой ближнее и дальнее фокусное расстояние соответсвенно.

Чтобы построить проекцию нужно задать точку, которая называется центром проекции. Проекции строятся с помощью проецирующих лучей или

проекторов, которые выходят из центра проекции. Проекторы пересекают плоскость, которая называется проекционной или картинной плоскостью, и затем проходят через каждую точку трехмерного объекта и образуют тем самым проекцию.

Поскольку поверхность любого трехмерного объекта содержит бесконечное число точек, то необходимо задать способ описания поверхности объекта конечным числом точек для представления в компьютере. А именно, будем использовать линейное представление объектов в трехмерном пространстве с помощью отрезков прямых и плоских многоугольников. При этом отрезки прямых после перспективного преобразования переходят в отрезки прямых на проекционной плоскости. Это важное свойство центральной перспективы позволяет проецировать, т.е. производить вычисления только для конечных точек отрезков, а затем соединять проекции точек линиями уже на проекционной плоскости.

```
141     public static MyMatrix4x4 CreateProjectionFOV(float A, float B, float C, float D, float E)
142     {
143         MyMatrix4x4 temp = new MyMatrix4x4();
144
145         temp.points[0, 0] = A;
146         temp.points[1, 1] = B;
147
148         temp.points[2, 2] = C;
149         temp.points[3, 2] = D;
150         temp.points[2, 3] = E;
151         temp.points[3, 3] = 0;
152
153         return temp;
154     }
```

Рисунок 4.8 – Формирование матрицы перспективной проекции

4.6 Реализация сохранения и открытия сцены

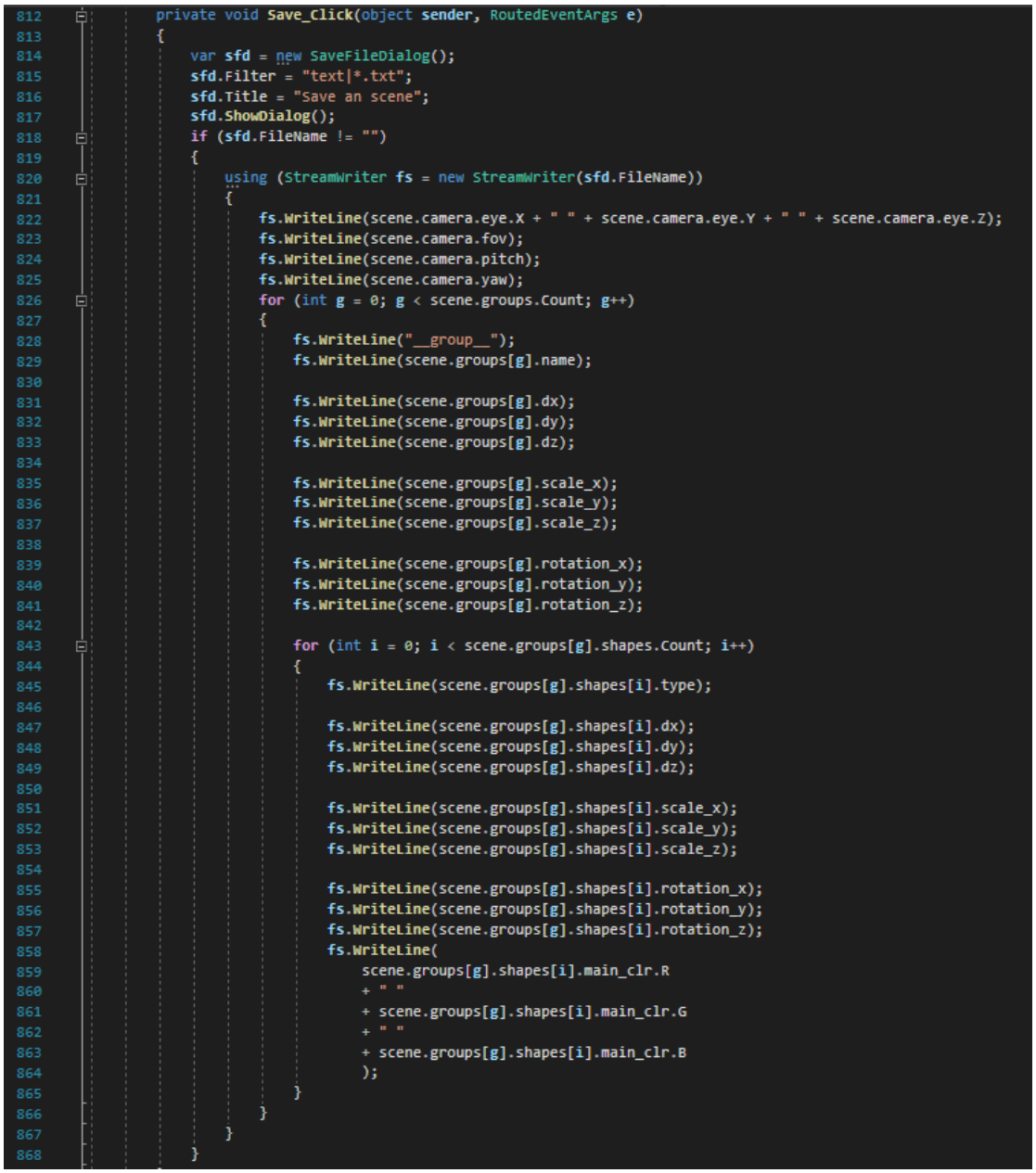
Объекты, добавленные пользователем на сцену храняться в динамическом списке. При сохранении сцены в файл происходит проход по всем элементам данного списка и запись параметров каждого объекта в файл.

Так же в файл сцены записывается текущее положение точки зрения и точки цели. Стуктура файла сцены привидена на рисунке 4.9.

<положение точки зрения>
<положение точки цели>
<параметры для матрицы модели объекта 1>
<параметры объекта1>
….
<параметры для матрицы модели объекта n>
<параметры объекта n>

Рисунок 4.9 – Структура хранимого файла сцены

При открытии файла сцены считывается количество объектов, далее в цикле читаются параметры каждого объекта и параметры его матрцицы модели. После это происходит расчет матриц модели каждого объекта и построение массива полигонов по его параметрам. Положение точки зрения и цели так же сохраняюся в поля объекта Scene.camera. Алгоритм записи сцены в файл и ее чтения привидены на рисунках 4.10 и 4.11 соответственно.



```
812         private void Save_Click(object sender, RoutedEventArgs e)
813         {
814             var sfd = new SaveFileDialog();
815             sfd.Filter = "text|*.txt";
816             sfd.Title = "Save an scene";
817             sfd.ShowDialog();
818             if (sfd.FileName != "")
819             {
820                 using (StreamWriter fs = new StreamWriter(sfd.FileName))
821                 {
822                     fs.WriteLine(scene.camera.eye.X + " " + scene.camera.eye.Y + " " + scene.camera.eye.Z);
823                     fs.WriteLine(scene.camera.fov);
824                     fs.WriteLine(scene.camera.pitch);
825                     fs.WriteLine(scene.camera.yaw);
826                     for (int g = 0; g < scene.groups.Count; g++)
827                     {
828                         fs.WriteLine("__group__");
829                         fs.WriteLine(scene.groups[g].name);
830 
831                         fs.WriteLine(scene.groups[g].dx);
832                         fs.WriteLine(scene.groups[g].dy);
833                         fs.WriteLine(scene.groups[g].dz);
834 
835                         fs.WriteLine(scene.groups[g].scale_x);
836                         fs.WriteLine(scene.groups[g].scale_y);
837                         fs.WriteLine(scene.groups[g].scale_z);
838 
839                         fs.WriteLine(scene.groups[g].rotation_x);
840                         fs.WriteLine(scene.groups[g].rotation_y);
841                         fs.WriteLine(scene.groups[g].rotation_z);
842 
843                         for (int i = 0; i < scene.groups[g].shapes.Count; i++)
844                         {
845                             fs.WriteLine(scene.groups[g].shapes[i].type);
846 
847                             fs.WriteLine(scene.groups[g].shapes[i].dx);
848                             fs.WriteLine(scene.groups[g].shapes[i].dy);
849                             fs.WriteLine(scene.groups[g].shapes[i].dz);
850 
851                             fs.WriteLine(scene.groups[g].shapes[i].scale_x);
852                             fs.WriteLine(scene.groups[g].shapes[i].scale_y);
853                             fs.WriteLine(scene.groups[g].shapes[i].scale_z);
854 
855                             fs.WriteLine(scene.groups[g].shapes[i].rotation_x);
856                             fs.WriteLine(scene.groups[g].shapes[i].rotation_y);
857                             fs.WriteLine(scene.groups[g].shapes[i].rotation_z);
858                             fs.WriteLine(
859                                 scene.groups[g].shapes[i].main_clr.R
860                                 + " "
861                                 + scene.groups[g].shapes[i].main_clr.G
862                                 + " "
863                                 + scene.groups[g].shapes[i].main_clr.B
864                                 );
865                         }
866                     }
867                 }
868             }
```

Рисунок 4.10 – Алгоритм записи сцены в файл

```
using (StreamReader fs = new StreamReader(lfd.FileName))
{
    string line;

    var eye = fs.ReadLine().Split(" ");
    scene.camera.eye = new Vector3(float.Parse(eye[0]), float.Parse(eye[1]), float.Parse(eye[2]));
    scene.camera.fov = float.Parse(fs.ReadLine());
    scene.camera.pitch = float.Parse(fs.ReadLine());
    scene.camera.yaw = float.Parse(fs.ReadLine());

    while ((line = fs.ReadLine()) != null)
    {
        if (line != "")
        {
            if(line == "__group__")
            {
                scene.groups.Add(new GroupShapes());
                scene.groups[scene.groups.Count - 1].name = fs.ReadLine();

                scene.groups[scene.groups.Count - 1].dx = float.Parse(fs.ReadLine());
                scene.groups[scene.groups.Count - 1].dy = float.Parse(fs.ReadLine());
                scene.groups[scene.groups.Count - 1].dz = float.Parse(fs.ReadLine());

                scene.groups[scene.groups.Count - 1].scale_x = float.Parse(fs.ReadLine());
                scene.groups[scene.groups.Count - 1].scale_y = float.Parse(fs.ReadLine());
                scene.groups[scene.groups.Count - 1].scale_z = float.Parse(fs.ReadLine());

                scene.groups[scene.groups.Count - 1].rotation_x = float.Parse(fs.ReadLine());
                scene.groups[scene.groups.Count - 1].rotation_y = float.Parse(fs.ReadLine());
                scene.groups[scene.groups.Count - 1].rotation_z = float.Parse(fs.ReadLine());
                continue;
            }
            Scene.Shape shape = new Scene.Shape();
            if (line == "Box")...
            else if (line == "IcoSphere")...
            else if (line == "Toroid")...
            else if (line == "Conus")...
            else if (line == "Cylinder")...

            shape.dx = float.Parse(fs.ReadLine());
            shape.dy = float.Parse(fs.ReadLine());
            shape.dz = float.Parse(fs.ReadLine());

            shape.scale_x = float.Parse(fs.ReadLine());
            shape.scale_y = float.Parse(fs.ReadLine());
            shape.scale_z = float.Parse(fs.ReadLine());

            shape.rotation_x = float.Parse(fs.ReadLine());
            shape.rotation_y = float.Parse(fs.ReadLine());
            shape.rotation_z = float.Parse(fs.ReadLine());

            var clr = fs.ReadLine().Split(" ");
            shape.main_clr = Render.color(int.Parse(clr[0]), int.Parse(clr[1]), int.Parse(clr[2]));
            scene.groups[scene.groups.Count-1].shapes.Add(shape);
        }
    }
    RefreshComboBox();
```

Рисунок 4.11 – Алгоритм чтения сцены из фалйа

5 ТЕСТИРОВАНИЕ ПРОГРАММЫ

Разработанная программная система отвечает всем требованиям технического задания, предоставляет пользователю возможность работы с полигональной моделью, возможность параметризации объекта, объединения объектов в группы, поворота, перемещения, масштабирования, удаления и видовых преобразований. Программа имеет интуитивно понятный пользовательский интерфейс на русском языке.

Разработанная программная система кроме исполнительных модулей предусматривает наличие модулей обработки исключительных ситуаций неверного задания параметров, а также неверного формата ввода чисел.

Тестирование программных модулей показало положительные результаты работы программы, как на корректных данных, так и с заведомо неверными. Программа запрещает пользователю вводить неверные данные, валидация данных проводится на каждой нажатой клавише.

В общих случаях тестирование производительности может служить разным целям.

С целью демонстрации того, что система удовлетворяет критериям производительности.

С целью определения, производительность какой из двух или нескольких систем лучше.

С целью определения, какой элемент нагрузки или часть системы приводит к снижению производительности.

Многие тесты на производительность делаются без попытки осмыслить их реальные цели. Перед началом тестирования всегда должен быть задан бизнес-вопрос: «Какую цель мы преследуем, тестируя производительность?». Ответы на этот вопрос являются частью технико-экономического обоснования (или business case) тестирования. Цели могут различаться в

зависимости от технологий, используемых приложением, или его назначения, однако, они всегда включают что-то из нижеследующего:

Параллелизм / Пропускная способность

Если конечными пользователями приложения считаются пользователи, выполняющие логин в систему в любой форме, то в этом случае крайне желательно достижение параллелизма. По определению это максимальное число параллельных работающих пользователей приложения, поддержка которого ожидается от приложения в любой момент времени. Модель поведения пользователя может значительно влиять на способность приложения к параллельной обработке запросов, особенно если он включает в себя периодически вход и выход из системы.

Если концепция приложения не заключается в работе с конкретными конечными пользователями, то преследуемая цель для производительности будет основана на максимальной пропускной способности или числе транзакций в единицу времени. Хорошим примером в данном случае будет являться просмотр веб-страниц, например, на портале Wikipedia.

Время отображения

Время отображения — одно из самых сложных для приложения для нагрузочного тестирования понятий, так как в общем случае они не используют концепцию работы с тем, что происходит на отдельных узлах системы, ограничиваясь только распознаванием периода времени, в течение которого нет сетевой активности. Для того, чтобы замерить время отображения, в общем случае требуется включать функциональные тестовые сценарии в тесты производительности, но большинство приложений для тестирования производительности не включают в себя такую возможность.

Требования к производительности

Очень важно детализировать требования к производительности и документировать их в каком-либо плане тестирования производительности. В идеальном случае это делается на стадии разработки требований при разработке системы, до проработки деталей её дизайна.

Однако тестирование производительности часто не проводится согласно спецификации, так как нет зафиксированного понимания о максимальном времени ответа для заданного числа пользователей. Тестирование производительности часто используется как часть процесса профайлинга производительности. Его идея заключается в том, чтобы найти «слабое звено» — такую часть системы, соптимизировав время реакции которой, можно улучшить общую производительность системы. Определение конкретной части системы, стоящей на этом критическом пути, иногда очень непростая задача, поэтому некоторые приложения для тестирования включают в себя (или могут быть добавлены с помощью add-on’ов) инструменты, запущенные на сервере (агенты) и наблюдающие за временем выполнения транзакций, временем доступа к базе данных, оверхедами сети и другими показателями серверной части системы, которые могут быть проанализированы вместе с остальной статистикой по производительности.

Тестирование производительности может проводиться с использованием глобальной сети и даже в географически удаленных местах, если учитывать тот факт, что скорость работы сети Интернет зависит от местоположения. Оно также может проводиться и локально, но в этом случае необходимо настроить сетевые маршрутизаторы таким образом, чтобы появилась задержка, присутствующая во всех публичных сетях. Нагрузка, прилагаемая к системе, должна совпадать с реальным положением дел. Так, например, если 50 % пользователей системы для доступа к системе используют сетевой канал шириной 56К, а другая половина использует оптический канал, то компьютеры, создающие тестовую нагрузку на систему

должны использовать те же соединения (идеальный вариант) или эмулировать задержки вышеуказанных сетевых соединений, следуя заданным профайлам пользователей.

ВЫВОДЫ

В ходе курсового проекта был создан графический редактор для работы с трехмерными обектами, преобразованиями над ними и камерой. Были получены и применены на практике знания в области 3D-моделирования и 3D-рендеринга.

Разработанный графический редактор отвечает требованиям, выдвинутым в техническом задании. Приложение реализует фунционал, необходимый для работы с трехмерным объектом и камерой.

ПЕРЕЧЕНЬ ССЫЛОК

1. COMGRAPH [Электронный ресурс] // Алгоритм, использующий z–буфер. – Режим доступа: http://compgraph.tpu.ru/zbuffer.htm (дата обращения 15.05.2016).

2. ESate [Электронный ресурс] // Трехмерная графика. – Режим доступа: http://esate.ru/article/cg/trekhmernaya_grafika/ (дата обращения 15.05.2016).

3. Википедия [Электронный ресурс] // Тестирование производительности – Режим доступа: https://ru.wikipedia.org/wiki/Тестирование_производительности .

ПРИЛОЖЕНИЕ А. ЭКРАННЫЕ ФОРМЫ

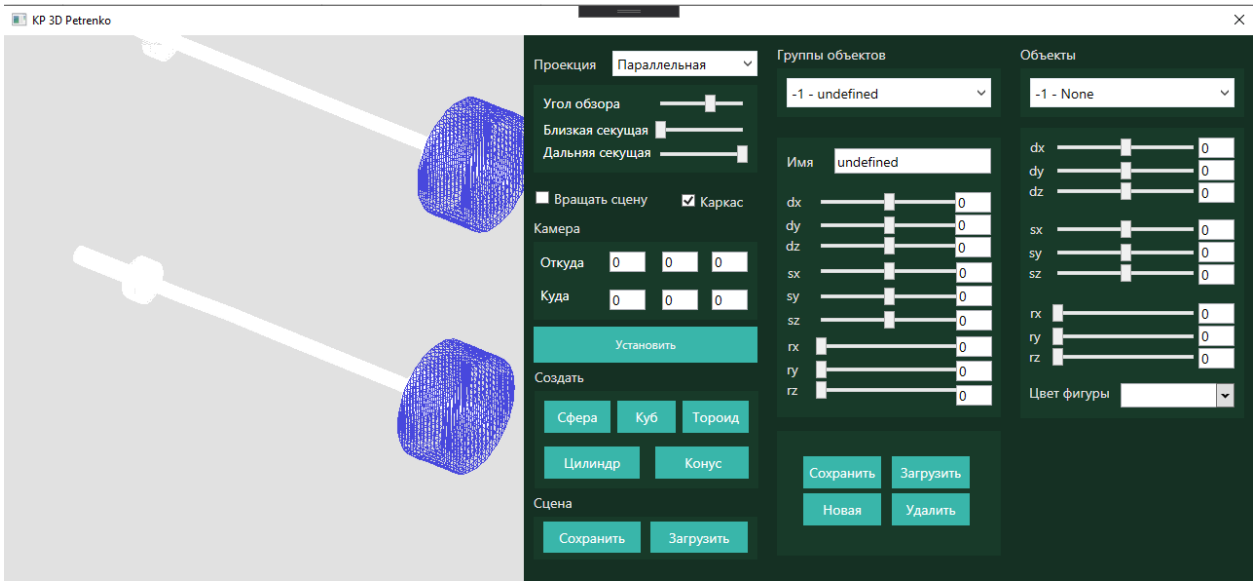


Рисунок А.1

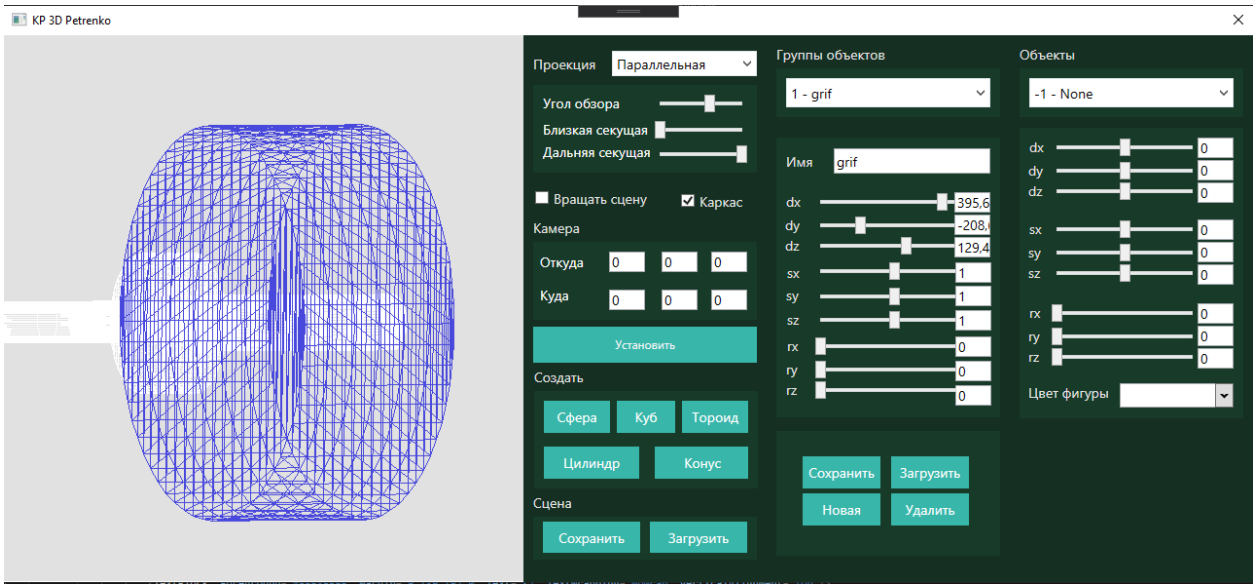


Рисунок А.2

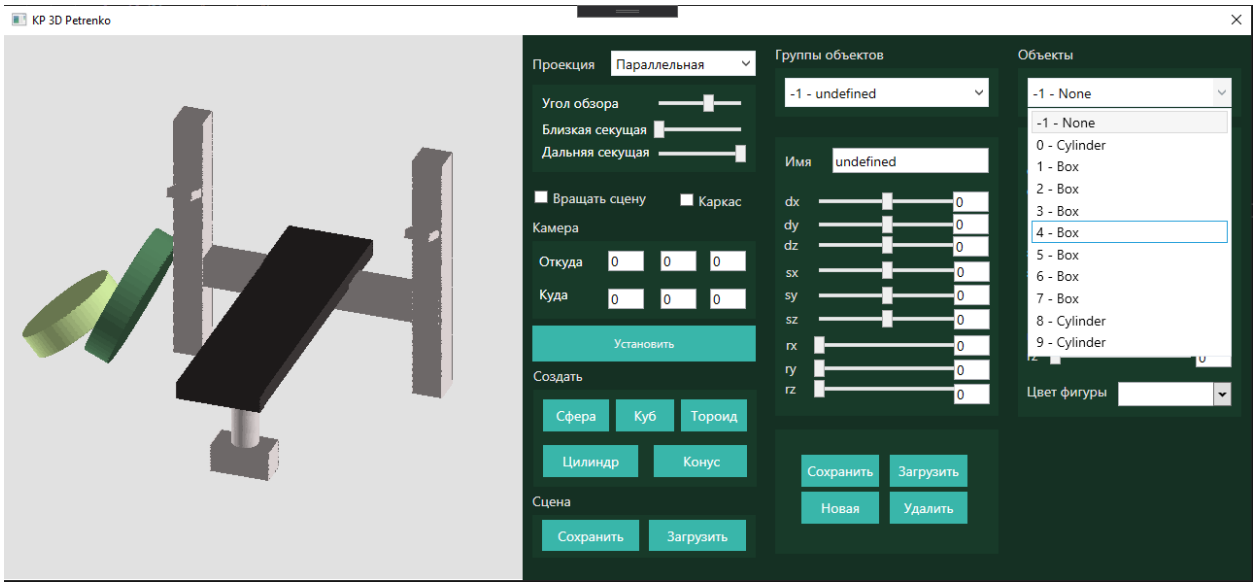


Рисунок А.3

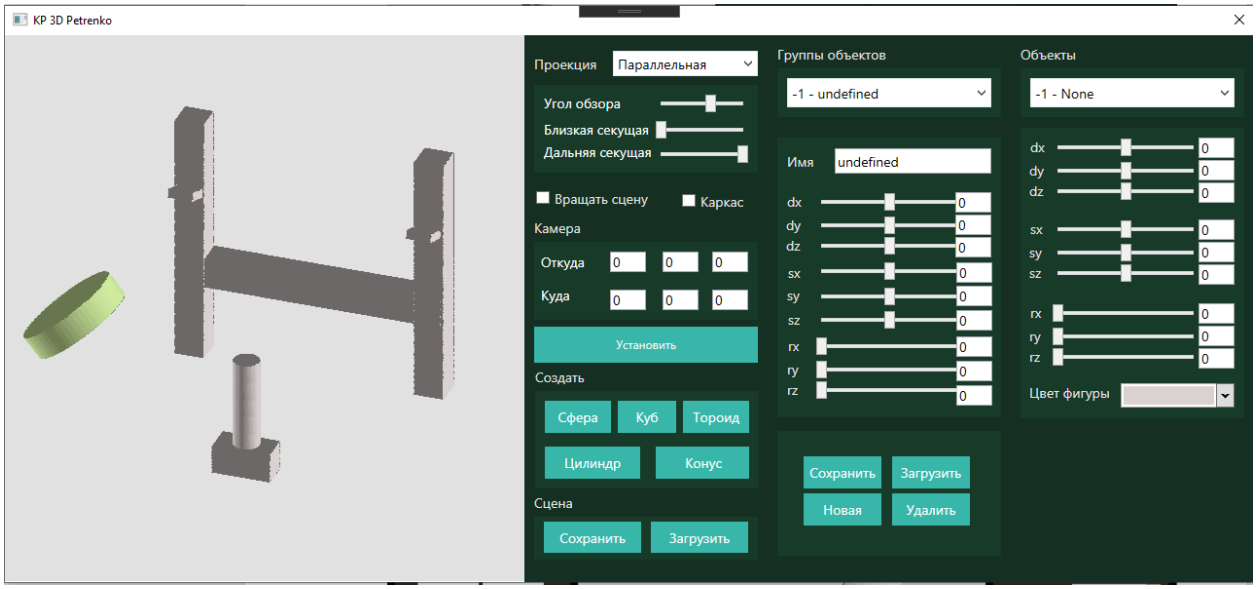


Рисунок А.4

ПРИЛОЖЕНИЕ Б. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

ГОУ ВПО «ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ НАУК И ТЕХНОЛОГИЙ

КАФЕДРА КМД

Дисциплина “Архитектура и проектирование графических систем”

Специальность «Программная инженерия»

Курс 3 Группа ПИ-17в Семестр 6

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

к курсовому проекту

по курсу «Архитектура и проектирование графических систем»

Петренко Д.А.

ТЕМА ПРОЕКТА: Разработка графического редактора для работы с параметризованными трёхмерными объектами

СРОК СДАЧИ:

ЗАДАНИЕ: Создать графический редактор для работы с трёхмерным объектом, изображённым на рисунке А.1.

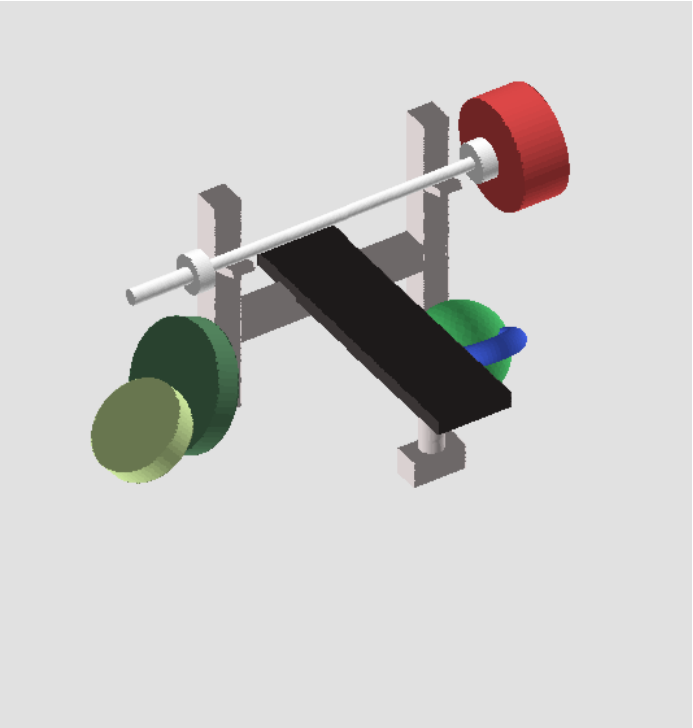


Рисунок А.1- Тренажер

Объект задается следующими параметрами:

- Длина лавки;
- Высота опоры;
- Радиус маленького блина;
- Радиус большого блина;
- Длина грифа;
- Радиус гантели;
- Ширина лавки;
- Ширина ручки гантели;
- Длина держателя для грифа;
- Высота держателя для грифа;
- Ширина маленького блина;
- Ширина большого блина.

ТРЕБОВАНИЯ К ГРАФИЧЕСКОМУ РЕДАКТОРУ

1. Наличие графической базы данных:

Возможность сохранения сцены с объектами в файле.

1.1 Читабельность базы данных:

Файл сцены должен содержать данные модели в текстовом виде.

1.2 Возможность работы с несколькими объектами:

Обеспечить добавление на экран допустимого количества объектов, а также работу со всеми объектами (перемещение, панорамирование) и одним выбранным объектом.

2. Обеспечить редактирование и параметризацию объектов:

Возможность изменения параметров любого объекта, а также его масштабирование, перенос, поворот и удаление.

3. Обеспечить центральное и параллельное проецирование:

Возможность переключения с одного вида проецирования на другой

4. Задание всех параметров аппарата проецирования:

Обеспечить наличие “камеры”, задаваемой необходимыми параметрами (как минимум – точка зрения и точка цели), также возможность её перемещения вокруг объекта и поворота вокруг своей оси.

5. Удаление невидимых частей объектов:

Обеспечить визуализацию объекта без его невидимых частей при помощи алгоритма удаления невидимых линий.

6. Разработать интуитивно понятный пользовательский интерфейс:

Программный продукт должен обеспечить пользователю максимально понятную и простую работу в редакторе за счёт оформления интерфейса, контекстных подсказок, горячих клавиш и предупреждений.

7. При разработке графического редактора не использовать стандартные графические библиотеки (Open GL, Direct X и т.п.).

СОДЕРЖАНИЕ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

- Разработка полигональной модели объекта
- Описание выбранных методов и алгоритмов визуализации
- Разработка структур данных для хранения описания объекта
- Программная реализация графического редактора
- Пример выполнения программы, иллюстрированный экранными формами

ДАТА ВЫДАЧИ ЗАДАНИЯ: 06.02.2020

Задание принял:

студент группы ПИ-17в _____ Петренко Д. А.

Руководители проекта:

зав. кафедрой КМД _____ Карабчевский В. В.

преподаватель _____ Бондар А. В.

доцент кафедры КМД _____ Доценко Г. В

ПРИЛОЖЕНИЕ В. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

В данной работе пользователь имеет возможность управления выбранным объектом и камерой. Операции с объектами производятся посредством интерфейса. В правой части окна присутствуют необходимые поля ввода координат обекта, его масштабирования и вращения. В левой части окна находится меню управления группами объектов. Для изменения положения положения объектов в мире достаточно покрутить ползунки. Для вращения камеры необходимо зажать ЛКМ на картинке и потянуть в сторону. Объекты можно выделять курсором мышки. Для удаления – клавиша Delete. Для передвижения камеры – WASD, UJ; Пример вращения камеры приведен на рисунке Б.1.

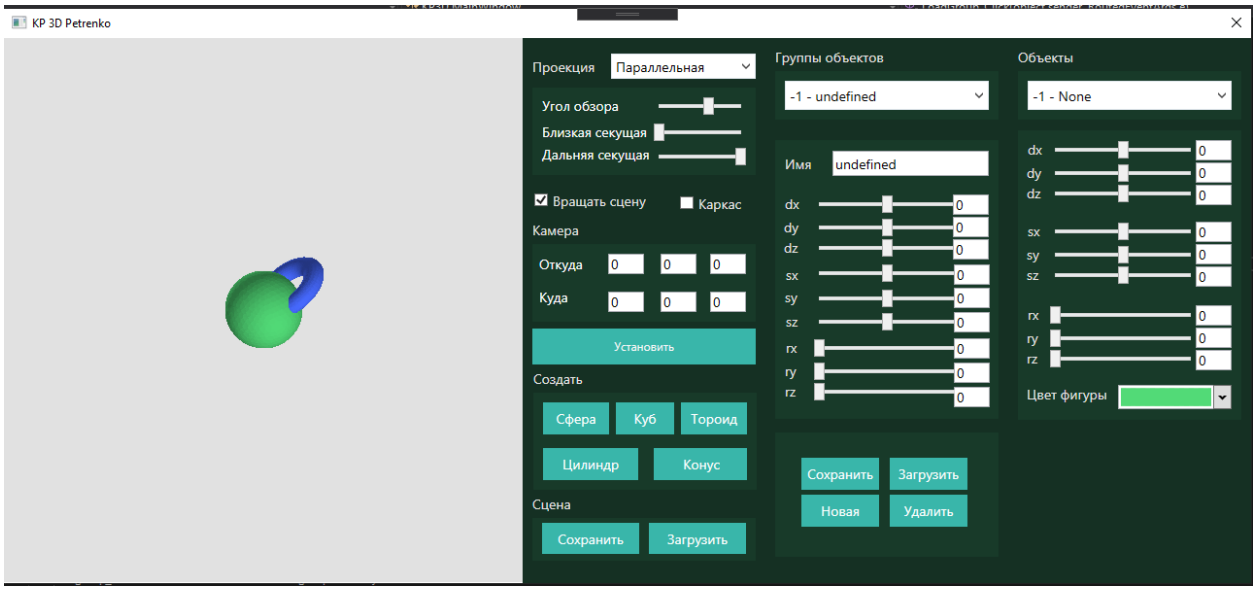


Рисунок Б.1 – Пример вращения объекта

Пример управления камерой приведен на рисунке Б.2.

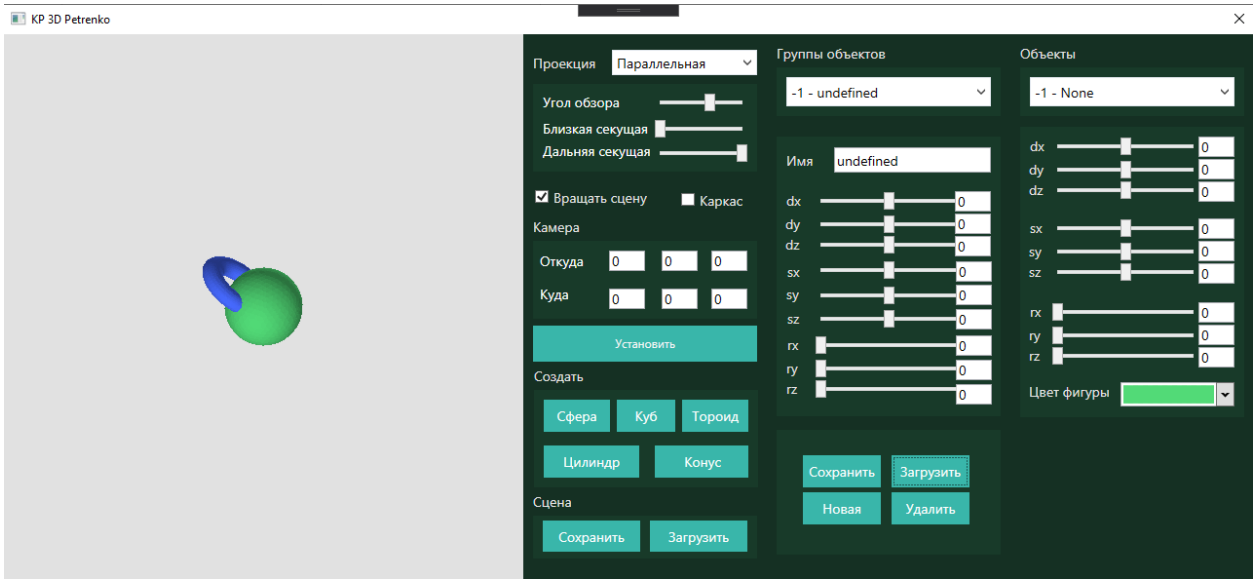


Рисунок Б.2 – Пример смещения камеры

ПРИЛОЖЕНИЕ Г. ЛИСТИНГ ПРОГРАММЫ

```
using Ara3D;
using System;
using
System.Collections.Generi
c;
using System.Diagnostics;
using System.Text;

namespace KP3D.MyMath
{
    class MyMatrix4x4
    {
        public float[,]
points = new float[4, 4];

        public
MyMatrix4x4()
        {
            points[0, 0]
= 1;
            points[1, 1]
= 1;
            points[2, 2]
= 1;
            points[3, 3]
= 1;
        }

        public static
MyMatrix4x4 Identity()
        {
            return new
MyMatrix4x4();
        }
        public static
MyMatrix4x4
CreateScale(float x,
float y, float z)
        {
            var temp =
new MyMatrix4x4();

temp.points[0, 0] = x;

temp.points[1, 1] = y;

temp.points[2, 2] = z;
            return temp;
        }

        public static
MyMatrix4x4
CreateTranslation(float
dx, float dy, float dz)
        {
            MyMatrix4x4
temp = new MyMatrix4x4();

temp.points[0, 3] = dx;

temp.points[1, 3] = dy;

temp.points[2, 3] = dz;
            return temp;
        }

        public static
MyMatrix4x4
CreateRotationX(float A)
        {
            MyMatrix4x4
temp = new MyMatrix4x4();

temp.points[1, 1] =
(float)Math.Cos(A);

temp.points[1, 2] =
(float)Math.Sin(A);

temp.points[2, 2] =
(float)Math.Cos(A);

temp.points[2, 1] = -
(float)Math.Sin(A);
            return temp;
        }

        public static
MyMatrix4x4
CreateRotationY(float A)
        {
            MyMatrix4x4
temp = new MyMatrix4x4();

temp.points[0, 0] =
(float)Math.Cos(A);

temp.points[2, 0] = -
(float)Math.Sin(A);

temp.points[0, 2] =
(float)Math.Sin(A);

temp.points[2, 2] =
(float)Math.Cos(A);
            return temp;
        }

        public static
MyMatrix4x4
CreateRotationZ(float A)
        {
            MyMatrix4x4
temp = new MyMatrix4x4();

temp.points[0, 0] =
(float)Math.Cos(A);

temp.points[1, 0] = -
(float)Math.Sin(A);

temp.points[0, 1] =
(float)Math.Sin(A);

temp.points[1, 1] =
(float)Math.Cos(A);
            return temp;
        }

        public static
MyMatrix4x4
CreateFPSLookAt(Vector3
eye, float pitch, float
yaw)
        {
            float
cosPitch =
(float)Math.Cos(pitch);
            float
sinPitch =
(float)Math.Sin(pitch);
            float cosYaw
= (float)Math.Cos(yaw);
            float sinYaw
= (float)Math.Sin(yaw);

            Vector3 xaxis
= new Vector3(cosYaw, 0,
-sinYaw);
            Vector3 yaxis
= new Vector3(sinYaw *
sinPitch, cosPitch,
cosYaw * sinPitch);
            Vector3 zaxis
= new Vector3(sinYaw *
cosPitch, -sinPitch,
cosPitch * cosYaw);

            MyMatrix4x4
view = new MyMatrix4x4();

view.points[0, 0] =
xaxis.X;

view.points[1, 0] =
yaxis.X;

view.points[2, 0] =
zaxis.X;
```

```
view.points[0, 1] =
xaxis.Y;

view.points[1, 1] =
yaxis.Y;

view.points[2, 1] =
zaxis.Y;


view.points[0, 2] =
xaxis.Z;

view.points[1, 2] =
yaxis.Z;

view.points[2, 2] =
zaxis.Z;


view.points[0, 3] = -
xaxis.Dot(eye);

view.points[1, 3] = -
yaxis.Dot(eye);

view.points[2, 3] = -
zaxis.Dot(eye);
            return
view;// *
MyMath.MyMatrix4x4.Create
Translation(eye.X, eye.Y,
eye.Z);
        }

        public static
MyMatrix4x4
CreateLookAt(Vector3 eye,
Vector3 center, Vector3
up)
        {
            Vector3 zaxis
= (eye -
center).Normalize();
// The "forward" vector.
            Vector3 xaxis
=
up.Cross(zaxis).Normalize
();   // The "right"
vector.
            Vector3 yaxis
= zaxis.Cross(xaxis);
// The "up" vector.
            MyMatrix4x4
view = new MyMatrix4x4();

view.points[0, 0] =
xaxis.X;

view.points[0, 1] =
xaxis.Y;

view.points[0, 2] =
xaxis.Z;

view.points[0, 3] =
xaxis.Dot(eye);


view.points[1, 0] =
yaxis.X;

view.points[1, 1] =
yaxis.Y;

view.points[1, 2] =
yaxis.Z;

view.points[1, 3] =
yaxis.Dot(eye);


view.points[2, 0] =
zaxis.X;

view.points[2, 1] =
zaxis.Y;

view.points[2, 2] =
zaxis.Z;

view.points[2, 3] =
zaxis.Dot(eye);
            return view;
        }

        public static
MyMatrix4x4
CreateProjection(float
left, float right, float
bottom, float top, float
near, float far)
        {
            MyMatrix4x4
temp = new MyMatrix4x4();

temp.points[0,0] = 2 *
near / (right - left);

temp.points[1,1] = 2 *
near / (top - bottom);

temp.points[2,2] = -(far
+ near) / (far - near);

temp.points[2,3] = -1;

temp.points[3,2] = -2 *
far * near / (far -
near);

temp.points[2,0] = (right
+ left) / (right - left);

temp.points[2,1] = (top +
bottom) / (top - bottom);

temp.points[3,3] = 0;

            return temp;
        }

        public static
MyMatrix4x4
CreateProjectionFOV(float
A, float B, float C,
float D, float E)
        {
            MyMatrix4x4
temp = new MyMatrix4x4();


temp.points[0, 0] = A;

temp.points[1, 1] = B;


temp.points[2, 2] = C;

temp.points[3, 2] = D;

temp.points[2, 3] = E;

temp.points[3, 3] = 0;

            return temp;
        }

        public static
MyMatrix4x4
CreateProjectionFOV(float
fov, float aspect, float
near, float far)
        {
            MyMatrix4x4
temp = new MyMatrix4x4();


temp.points[0, 0] = 1 /
(float)Math.Tan(fov *
0.5f) / aspect;

temp.points[1, 1] = 1 /
(float)Math.Tan(fov *
0.5f);


temp.points[2, 2] = (far)
/ (far - near);

temp.points[3, 2] = -1f;

temp.points[2, 3] = -near
* far / (far - near);

temp.points[3, 3] = 0;
```

```
            //var a =
Matrix4x4.CreatePerspecti
veFieldOfView(fov,
aspect, near, far);
            //temp.points
= new float[,] { { a.M11,
a.M12, a.M13, a.M14 }, {
a.M21, a.M22, a.M23,
a.M24 }, { a.M31, a.M32,
a.M33, a.M34 }, { a.M41,
a.M42, a.M43, a.M44 } };
            //temp.points
= new float[,] { { a.M11,
a.M21, a.M31, a.M41 }, {
a.M12, a.M22, a.M32,
a.M42 }, { a.M13, a.M23,
a.M33, a.M43 }, { a.M14,
a.M24, a.M34, a.M44 } };

            return temp;
        }

        public static
MyMatrix4x4 operator
*(MyMatrix4x4 A,
MyMatrix4x4 B)
        {
            MyMatrix4x4 C
= new MyMatrix4x4();

            for (var i =
0; i < 4; i++)
            {
                for (var
j = 0; j < 4; j++)
                {

C.points[i, j] = 0;

                    for
(var k = 0; k < 4; k++)
                    {

C.points[i, j] +=
A.points[i, k] *
B.points[k, j];
                    }
                }
            }

            return C;
        }

        public static
MyMatrix4x4
FromPoint(Vector3 vector)
        {
            MyMatrix4x4
temp = new MyMatrix4x4();

temp.points[0, 0] =
vector.X;

temp.points[1, 1] =
vector.Y;

temp.points[2, 2] =
vector.Z;
            return temp;
        }

        public static
Vector3
ToPoint(MyMatrix4x4 m)
        {
            return new
Vector3(m.points[0, 0],
m.points[1, 1],
m.points[2, 2]);
        }

        public static
MyMatrix4x4
inverse(MyMatrix4x4
matrix)
        {
            return null;
        }
    }
    class MyMatrix4x1
    {
        public float[,]
points = new float[4, 1];

        public
MyMatrix4x1()
        {
            points[0, 0]
= 1;
            points[1, 0]
= 1;
            points[2, 0]
= 1;
            points[3, 0]
= 1;
        }

        public static
MyMatrix4x1 operator
*(MyMatrix4x4 A,
MyMatrix4x1 B)
        {
            MyMatrix4x1 C
= new MyMatrix4x1();

            for (var i =
0; i < 4; i++)
            {


C.points[i, 0] = 0;

                for (var
k = 0; k < 4; k++)
                {

C.points[i, 0] +=
A.points[i, k] *
B.points[k, 0];
                }
            }

            return C;
        }

        public static
MyMatrix4x1
FromPoint4x1(Vector3
vector)
        {
            MyMatrix4x1
temp = new MyMatrix4x1();

temp.points[0, 0] =
vector.X;

temp.points[1, 0] =
vector.Y;

temp.points[2, 0] =
vector.Z;
            return temp;
        }
        public static
Vector3
ToPoint(MyMatrix4x1 m)
        {
            float x =
m.points[0, 0];
            float y =
m.points[1, 0];
            float z =
m.points[2, 0];

            return new
Vector3(x, y, z);
        }

        public static
Vector3
ToPoint(MyMatrix4x1 m,
float A, float B, float
C, float D, float E, int
width, int height)
        {
            float x =
m.points[0, 0];
            float y =
m.points[1, 0];
            float z =
m.points[2, 0];
            float w =
m.points[3, 0];

            //x = (A * x)
/ (D * z);
            //y = (B * y)
/ (D * z);
```

```
            //x = (x) *
width;
            //y = (y) *
height;


            //z = (C * z
+ E * w * E) / (D * z);
            //z =
MathF.Log2(MathF.Max(1e-
6f, 1.0f + w)) * 2.0f /
MathF.Log2(100f + 1.0f) -
1.0f;
            //z =
MathF.Log2(1 * z + 1) /
MathF.Log2(1 * 100f + 1)
* w;

//Debug.WriteLine(z);

            return new
Vector3(x, y, z);
        }

        public static
Vector3
ToPointZisW(MyMatrix4x1
m)
        {
            float x =
m.points[0, 0];
            float y =
m.points[1, 0];
            float z =
m.points[2, 0];

            return new
Vector3(x, y, z /
m.points[3, 0]);
        }
    }

}
using Ara3D;
using KP3D.Shapes;
using System;
using
System.Collections.Generi
c;
using System.Diagnostics;
using System.Drawing;
using
System.Drawing.Imaging;
using
System.Runtime.InteropSer
vices;
using System.Text;
using
System.Threading.Tasks;

namespace KP3D.Scene
{
    class Scene
    {
        public Camera
camera;
        //public
List<Shape> shapes;
        public
List<GroupShapes> groups;

        public float[]
zbuffer;
        public int[]
id_object_in_pixel;
        public int[]
id_group_in_pixel;

        public int width;
        public int
height;

        public int
selected_id = -1;
        public int
selected_gid = -1;

        public
Scene(Camera camera)
        {
            this.camera =
camera;
            //this.shapes
= new List<Shape>();
            this.groups =
new List<GroupShapes>();
        }

        public
System.Drawing.Bitmap
render(int width, int
height, bool is_lines)
        {
            this.width =
width;
            this.height =
height;
            Bitmap bm =
new Bitmap(width,
height);

            BitmapData
bitmapData =
bm.LockBits(new
Rectangle(0, 0, bm.Width,
bm.Height),
ImageLockMode.ReadWrite,
bm.PixelFormat);
            int
bytesPerPixel =
Bitmap.GetPixelFormatSize
(bm.PixelFormat) / 8;
            int byteCount
= bitmapData.Stride *
bm.Height;
            byte[] pixels
= new byte[byteCount];
            IntPtr
ptrFirstPixel =
bitmapData.Scan0;

Marshal.Copy(ptrFirstPixe
l, pixels, 0,
pixels.Length);
            int
heightInPixels =
bitmapData.Height;
            int
widthInBytes =
bitmapData.Width *
bytesPerPixel;

            for (int y =
0; y < heightInPixels;
y++)
            {
                int
currentLine = y *
bitmapData.Stride;
                for (int
x = 0; x < widthInBytes;
x = x + bytesPerPixel)
                {

pixels[currentLine + x] =
(byte)225;

pixels[currentLine + x +
1] = (byte)225;

pixels[currentLine + x +
2] = (byte)225;
                }
            }

            zbuffer = new
float[width * height];

id_object_in_pixel = new
int[width * height];

id_group_in_pixel = new
int[width * height];
            for (int j =
0; j < width * height;
j++)
            {

zbuffer[j] = -0xFFFFFFFF;

id_object_in_pixel[j] = -
1;

id_group_in_pixel[j] = -
1;
            }

            for (int g =
0; g < groups.Count; g++)
            {
```

```
                float
rotGX =
groups[g].rotation_x;
                float
rotGY =
groups[g].rotation_y;
                float
rotGZ =
groups[g].rotation_z;

                float
scaleGX =
groups[g].scale_x;
                float
scaleGY =
groups[g].scale_y;
                float
scaleGZ =
groups[g].scale_z;

                float Gdx
= groups[g].dx;
                float Gdy
= groups[g].dy;
                float Gdz
= groups[g].dz;

                for (int
i = 0; i <
groups[g].shapes.Count;
i++)
                {

                    var
RotX =
MyMath.MyMatrix4x4.Create
RotationX(groups[g].shape
s[i].rotation_x);
                    var
RotY =
MyMath.MyMatrix4x4.Create
RotationY(groups[g].shape
s[i].rotation_y);
                    var
RotZ =
MyMath.MyMatrix4x4.Create
RotationZ(groups[g].shape
s[i].rotation_z);

                    var
RotXG =
MyMath.MyMatrix4x4.Create
RotationX(rotGX);
                    var
RotYG =
MyMath.MyMatrix4x4.Create
RotationY(rotGY);
                    var
RotZG =
MyMath.MyMatrix4x4.Create
RotationZ(rotGZ);

                    var
Rot = RotX * RotY * RotZ;
                    var
RotG = RotXG * RotYG *
RotZG;


                    var
Scale =
MyMath.MyMatrix4x4.Create
Scale(

groups[g].shapes[i].scale
_x * scaleGX,

groups[g].shapes[i].scale
_y * scaleGY,

groups[g].shapes[i].scale
_z * scaleGZ

);

                    var
TL =
MyMath.MyMatrix4x4.Create
Translation(

(groups[g].shapes[i].dx +
Gdx) * scaleGX,

(groups[g].shapes[i].dy +
Gdy) * scaleGY,

(groups[g].shapes[i].dz +
Gdz) * scaleGZ

);


                    var
TLG =
MyMath.MyMatrix4x4.Create
Translation(
                        -
(groups[g].shapes[i].dx +
Gdx) * scaleGX,
                        -
(groups[g].shapes[i].dy +
Gdy) * scaleGY,
                        -
(groups[g].shapes[i].dz +
Gdz) * scaleGZ

);


                    var W
=  TL * TLG * RotG * TL *
Rot * Scale;

                    var V
=
MyMath.MyMatrix4x4.Create
FPSLookAt(camera.eye,
camera.pitch,
camera.yaw);
                    if
(camera.center !=
Vector3.MinValue)
                        V
=
MyMath.MyMatrix4x4.Create
LookAt(camera.eye,
camera.center, new
Vector3(0, 1, 0));

                    float
far = 100;
                    float
near = 0.01f;

                    float
A = 1 /
(float)Math.Tan(camera.fo
v * 0.5f);
                    float
B = 1 /
(float)Math.Tan(camera.fo
v * 0.5f);
                    float
C = far / (far - near);
                    float
D = -1;
                    float
E = near * far / (far -
near);

                    //var
P =
MyMath.MyMatrix4x4.Create
ProjectionFOV(camera.fov,
(float)width /
(float)height, near,
far);
                    var P
=
MyMath.MyMatrix4x4.Create
ProjectionFOV(A, B, C, D,
E);

Object _lock = new
Object();

Parallel.For(0,
groups[g].shapes[i].trian
gles.Count, new
ParallelOptions {
MaxDegreeOfParallelism =
16 }, j =>
                    //for
(int j = 0; j <
shapes[i].triangles.Count
; j++)
                    {

var p0 =
MyMath.MyMatrix4x1.FromPo
int4x1(groups[g].shapes[i
].triangles[j].a);
```

```
var p1 =
MyMath.MyMatrix4x1.FromPo
int4x1(groups[g].shapes[i
].triangles[j].b);

var p2 =
MyMath.MyMatrix4x1.FromPo
int4x1(groups[g].shapes[i
].triangles[j].c);


var t1 =
MyMath.MyMatrix4x1.ToPoin
t(P * V * W * p0, A, B,
C, D, E, width, height);

var t2 =
MyMath.MyMatrix4x1.ToPoin
t(P * V * W * p1, A, B,
C, D, E, width, height);

var t3 =
MyMath.MyMatrix4x1.ToPoin
t(P * V * W * p2, A, B,
C, D, E, width, height);



var intensivity = 0.5 +
0.5 *
Math.Abs(groups[g].shapes
[i].triangles[j].normal.N
ormalize().Z);

if (intensivity.IsNaN())

intensivity = 0.5;



var color =
Render.color(255, 255,
255);


if (i == selected_id && g
== selected_gid)
                        {

color =
groups[g].shapes[i].selec
t_clr;
                        }

else
                        {

color =
groups[g].shapes[i].main_
clr;
                        }

var clr = Render.color(

(int)(color.B *
intensivity),

(int)(color.G *
intensivity),

(int)(color.R *
intensivity)

);

lock (_lock)
                            {

if (!is_lines)

{

Render.AndYetAnotherMemes
SavedTheWorld(t1, t2, t3,
clr, zbuffer, pixels,
bitmapData,
bytesPerPixel,
id_object_in_pixel, i,
id_group_in_pixel, g);

}

else

{

Render.DrawLine((int)t1.X
+ width / 2, (int)t1.Y +
height / 2, (int)t1.Z,
(int)t2.X + width / 2,
(int)t2.Y + height / 2,
t2.Z, color, pixels,
bitmapData,
bytesPerPixel, null,
id_object_in_pixel, i,
id_group_in_pixel, g);

Render.DrawLine((int)t1.X
+ width / 2, (int)t1.Y +
height / 2, (int)t1.Z,
(int)t3.X + width / 2,
(int)t3.Y + height / 2,
t3.Z, color, pixels,
bitmapData,
bytesPerPixel, null,
id_object_in_pixel, i,
id_group_in_pixel, g);

Render.DrawLine((int)t3.X
+ width / 2, (int)t3.Y +
height / 2, (int)t3.Z,
(int)t2.X + width / 2,
(int)t2.Y + height / 2,
t2.Z, color, pixels,
bitmapData,
bytesPerPixel, null,
id_object_in_pixel, i,
id_group_in_pixel, g);

}
                                }

                        }
                        );
                    }
                }

Marshal.Copy(pixels, 0,
ptrFirstPixel,
pixels.Length);

bm.UnlockBits(bitmapData)
;
            return bm;

        }

    }
}

using Ara3D;
using System;
using
System.Collections.Generi
c;
using System.Drawing;
using
System.Drawing.Imaging;
using System.Text;

namespace KP3D.Scene
{
    class Render
    {
        public static
System.Drawing.Color
color(int r, int g, int
b)
        {
            return
System.Drawing.Color.From
Argb(255, r, g, b);
        }
        static void
Swap<T>(ref T lhs, ref T
rhs)
        {
            T temp;
            temp = lhs;
            lhs = rhs;
            rhs = temp;
        }

        public static
void DrawLine(int x0, int
y0, double z0, int x1,
int y1, double z1,
System.Drawing.Color clr,
byte[] pixels, BitmapData
```

```
bd, int BytesPerPixel,
float[] zBuffer, int[]
id_object_in_pixel, int
id, int[]
id_group_in_pixel, int
gid)
        {
            var steep =
Math.Abs(y1 - y0) >
Math.Abs(x1 - x0);
            if (steep)
            {
                int t;
                t = x0;
                x0 = y0;
                y0 = t;
                t = x1;
                x1 = y1;
                y1 = t;
            }
            if (x0 > x1)
            {
                int t;
                t = x0;
                x0 = x1;
                x1 = t;
                t = y0;
                y0 = y1;
                y1 = t;
                Swap(ref
z0, ref z1);
            }
            var dx = x1 -
x0;
            var dy =
Math.Abs(y1 - y0);
            var error =
dx / 2;
            var ystep =
y0 < y1 ? 1 : -1;
            var y = y0;

            double deltaZ
= (z1 - z0) / (x1 - x0);
            double z =
z0;

            for (var x =
x0; x <= x1; x++)
            {
                if (x >=
bd.Width)

return;

                if (x >
0)
                {
                    bool
shouldBeDrawn = false;

                    if
(zBuffer == null)
                    {

//don't use Z buffer

shouldBeDrawn = true;
                    }
                    else
                    {

//check Z buffer

int index = x + y *
bd.Width;

if (index <
zBuffer.Length && index >
0)
                        {

if (z > zBuffer[index])

{

zBuffer[index] =
(float)z;

shouldBeDrawn = true;

if(id_object_in_pixel !=
null)

id_object_in_pixel[index]
= id;

if(id_group_in_pixel !=
null)

id_group_in_pixel[index]
= gid;

}
                        }
                    }

                    if
(shouldBeDrawn)

SetPixel(steep ? y : x,
steep ? x : y, clr,
pixels, bd,
BytesPerPixel);
                }


                error =
error - dy;
                if (error
< 0)
                {
                    y +=
ystep;
                    error
+= dx;
                }
                z +=
deltaZ;
            }
        }

        public static
void SetPixel(int x, int
y, System.Drawing.Color
clr, byte[] pixels,
BitmapData bd, int
BytesPerPixel)
        {
            if (x >= 0 &&
y >= 0 && x < bd.Width &&
y < bd.Height)
            {
                int
currentLine = (int)y *
bd.Stride;


pixels[currentLine +
(int)x * BytesPerPixel] =
(byte)(clr.R);

pixels[currentLine +
(int)x * BytesPerPixel +
1] = (byte)(clr.G);

pixels[currentLine +
(int)x * BytesPerPixel +
2] = (byte)(clr.B);
            }
        }

        public static
void
AndYetAnotherMemesSavedTh
eWorld(Vector3 v1,
Vector3 v2, Vector3 v3,
System.Drawing.Color clr,
float[] zBuffer, byte[]
pixels, BitmapData bd,
int BytesPerPixel, int[]
id_object_in_pixel, int
id, int[]
id_group_in_pixel, int
gid)
        {
            Vector2 min =
new
Vector2(Math.Min(v1.X,
Math.Min(v2.X, v3.X)),
Math.Min(v1.Y,
Math.Min(v2.Y, v3.Y)));
            Vector2 max =
new
Vector2(Math.Max(v1.X,
Math.Max(v2.X, v3.X)),
Math.Max(v1.Y,
Math.Max(v2.Y, v3.Y)));

            //min =
min.Clamp(Vector2.Zero,
```

```
new Vector2(bd.Width,
bd.Height));
            //max =
max.Clamp(Vector2.Zero,
new Vector2(bd.Width,
bd.Height));

            Vector3[]
verts = new Vector3[3] {
v1, v2, v3 };

            for (int yp =
(int)min.Y; yp <=
(int)max.Y; yp++)
            {

List<Vector3>
intersections = new
List<Vector3>();

                //voor
alle lijnen van de
triangle, vind alle
intersecties (als het
goed is altijd 2)
                for (int
l = 0; l < 3; l++)
                {

Vector3 vert1, vert2;
                    vert1
= verts[l];
                    vert2
= verts[(l + 1) % 3];

                    float
ymin, ymax;
                    ymin
= Math.Min(vert1.Y,
vert2.Y);
                    ymax
= Math.Max(vert1.Y,
vert2.Y);
                    if
(yp > ymin && yp <
ymax)//check of yp tussen
de y-en van de vertices
ligt, zo ja, intersectie
                    {

float xSlope = (vert2.X -
vert1.X) / (vert2.Y -
vert1.Y);

float xIntersect =
vert1.X + xSlope * (yp -
vert1.Y);


float zSlope = (vert2.Z -
vert1.Z) / (vert2.Y -
vert1.Y);

float zIntersect =
vert1.Z + zSlope * (yp -
vert1.Y);

intersections.Add(new
Vector3((int)xIntersect+b
d.Width/2,
yp+bd.Height/2,
zIntersect));
                    }
                }

                if
(intersections.Count > 1)
                {

Vector3 pStart, pEnd;
                    if
(intersections[0].X <
intersections[1].X) {
pStart =
intersections[0]; pEnd =
intersections[1]; }
                    else
                    {
pStart =
intersections[1]; pEnd =
intersections[0]; }


DrawLine((int)pStart.X,
(int)pStart.Y, pStart.Z,
(int)pEnd.X, (int)pEnd.Y,
pEnd.Z, clr, pixels, bd,
BytesPerPixel, zBuffer,
id_object_in_pixel, id,
id_group_in_pixel, gid);
                }

            }
        }
    }
}

using Ara3D;
using System;
using
System.Collections.Generi
c;
using System.Text;

namespace KP3D.Shapes
{
    class Box :
Scene.Shape
    {
        public Box()
        {
            type = "Box";
            triangles =
new List<MyTriangle>();

ToTriangles();
        }

        public void
ToTriangles()
        {
            //нижняя
грань

triangles.Add(new
MyTriangle(
                new
Vector3(-0.5f, -0.5f, -
0.5f),
                new
Vector3(0.5f, -0.5f, -
0.5f),
                new
Vector3(0.5f, 0.5f, -
0.5f)
                )
            );

triangles.Add(new
MyTriangle(
                new
Vector3(-0.5f, -0.5f, -
0.5f),
                new
Vector3(-0.5f, 0.5f, -
0.5f),
                new
Vector3(0.5f, 0.5f, -
0.5f)
                )
            );

            //верхняя
грань

triangles.Add(new
MyTriangle(
                new
Vector3(-0.5f, -0.5f,
0.5f),
                new
Vector3(0.5f, -0.5f,
0.5f),
                new
Vector3(0.5f, 0.5f, 0.5f)
                )
            );

triangles.Add(new
MyTriangle(
                new
Vector3(-0.5f, 0.5f,
0.5f),
                new
Vector3(-0.5f, -0.5f,
0.5f),
                new
Vector3(0.5f, 0.5f, 0.5f)
                )
            );

            //левая грань
```

```
triangles.Add(new
MyTriangle(
                new
Vector3(-0.5f, 0.5f, -
0.5f),
                new
Vector3(-0.5f, -0.5f, -
0.5f),
                new
Vector3(-0.5f, 0.5f,
0.5f)
                )
            );

triangles.Add(new
MyTriangle(
                new
Vector3(-0.5f, 0.5f,
0.5f),
                new
Vector3(-0.5f, -0.5f,
0.5f),
                new
Vector3(-0.5f, -0.5f, -
0.5f)
                )
           );
            //правая
грань

triangles.Add(new
MyTriangle(
                new
Vector3(0.5f, 0.5f, -
0.5f),
                new
Vector3(0.5f, 0.5f,
0.5f),
                new
Vector3(0.5f, -0.5f, -
0.5f)
                )
           );

triangles.Add(new
MyTriangle(
                new
Vector3(0.5f, 0.5f,
0.5f),
                new
Vector3(0.5f, -0.5f,
0.5f),
                new
Vector3(0.5f, -0.5f, -
0.5f)
                )
           );
            //передняя
грань

triangles.Add(new
MyTriangle(
                new
Vector3(-0.5f, 0.5f, -
0.5f),
                new
Vector3(-0.5f, 0.5f,
0.5f),
                new
Vector3(0.5f, 0.5f, 0.5f)
                )
            );

triangles.Add(new
MyTriangle(
                new
Vector3(-0.5f, 0.5f, -
0.5f),
                new
Vector3(0.5f, 0.5f, -
0.5f),
                new
Vector3(0.5f, 0.5f, 0.5f)
                )
            );
            //задняя
грань

triangles.Add(new
MyTriangle(
                new
Vector3(-0.5f, -0.5f, -
0.5f),
                new
Vector3(-0.5f, -0.5f,
0.5f),
                new
Vector3(0.5f, -0.5f, -
0.5f)
                )
            );


triangles.Add(new
MyTriangle(
                new
Vector3(-0.5f, -0.5f,
0.5f),
                new
Vector3(0.5f, -0.5f,
0.5f),
                new
Vector3(0.5f, -0.5f, -
0.5f)
                )
            );
            for (int i =
0; i < triangles.Count;
i++)
            {
                float _a
= triangles[i].a.X +
triangles[i].b.X +
triangles[i].c.X;
                float _b
= triangles[i].a.Y +
triangles[i].b.Y +
triangles[i].c.Y;
                float _c
= triangles[i].a.Z +
triangles[i].b.Z +
triangles[i].c.Z;
                Vector3 v
= new Vector3(_a/3, _b/3,
_c/3);


triangles[i].normal =
(triangles[i].c -
v).Cross(triangles[i].a -
v);
            }
        }
    }
}

using Ara3D;
using System;
using
System.Collections.Generi
c;
using System.Text;

namespace KP3D.Shapes
{
    class Conus :
Scene.Shape
    {
        public Conus(int
subdivisions=60, float
radius=1, float height=2)
        {
            type =
"Conus";
            triangles =
new List<MyTriangle>();

ToTriangles(subdivisions,
radius, height);
        }
        public void
ToTriangles(int
subdivisions, float
radius, float height)
        {
            Vector3[]
vertices = new
Vector3[subdivisions +
2];


       int[] itriangles =
new int[(subdivisions *
2) * 3];


       vertices[0] =
Vector3.Zero;
```

```
                for (int i = 0, n = subdivisions - 1; i < subdivisions; i++)
                {

                    float ratio = (float)i / n;

                    float r = ratio * ((float)Math.PI * 2f);

                    float x = (float)Math.Cos(r) * radius;

                    float z = (float)Math.Sin(r) * radius;

                    vertices[i + 1] = new Vector3(x, 0f, z);

                }

                vertices[subdivisions + 1] = new Vector3(0f, height, 0f);

                // construct bottom

                for (int i = 0, n = subdivisions - 1; i < n; i++)
                {

                    int offset = i * 3;

                    itriangles[offset] = 0;

                    itriangles[offset + 1] = i + 1;

                    itriangles[offset + 2] = i + 2;
                }

                // construct sides

                int bottomOffset = subdivisions * 3;
                for (int i = 0, n = subdivisions - 1; i < n; i++)
                {

                    int offset = i * 3 + bottomOffset;

                    itriangles[offset] = i + 1;

                    itriangles[offset + 1] = subdivisions + 1;

                    itriangles[offset + 2] = i + 2;
                }

                for (int i = 0; i < itriangles.Length; i+=3)
                {

                    triangles.Add(new Shapes.MyTriangle(vertices[itriangles[i]], vertices[itriangles[i+1]], vertices[itriangles[i+2]]));
                }

                for (int i = 0; i < triangles.Count; i++)
                {

                    float _a = triangles[i].a.X + triangles[i].b.X + triangles[i].c.X;

                    float _b = triangles[i].a.Y + triangles[i].b.Y + triangles[i].c.Y;

                    float _c = triangles[i].a.Z + triangles[i].b.Z + triangles[i].c.Z;

                    Vector3 v = new Vector3(_a / 3, _b / 3, _c / 3);


                    triangles[i].normal = (triangles[i].c - v).Cross(triangles[i].a - v);


                    if (triangles[i].normal.Y < 0) //or whatever direction up is
                    {
                        triangles[i].normal = -triangles[i].normal;
                    }

                }
            }
        }
    }
}

using Ara3D;
using System;
using System.Collections.Generic;
using System.Text;

namespace KP3D.Shapes
{
    class Cylinder : Scene.Shape
    {
        public Cylinder()
        {
            type = "Cylinder";

            triangles = new List<MyTriangle>();

            ToTriangles();
        }
        public void ToTriangles()
        {

            float radius = 1.0f;

            float length = 2.0f;

            int radialSegments = 60;
            int heightSegments = 6;

            int numVertexColumns = radialSegments + 1;  //+1 for welding
            int numVertexRows = heightSegments + 1;

            int numVertices = numVertexColumns * numVertexRows;

            int numSideTris = radialSegments *
```

```
heightSegments * 2;
//for one cap
                    int
numCapTris =
radialSegments - 2;
//fact

                    int
trisArrayLength =
(numSideTris + numCapTris
* 2) * 3;   //3 places in
the array for each tri


      //initialize
arrays

      Vector3[] Vertices
= new
Vector3[numVertices];


      int[] Tris = new
int[trisArrayLength];


      //precalculate
increments to improve
performance

      float heightStep =
length / heightSegments;

      float angleStep =
2 * (float)Math.PI /
radialSegments;

                    for
(int j = 0; j <
numVertexRows; j++)
                    {

      for (int i = 0; i
< numVertexColumns; i++)

      {

            //calculate
angle for that vertex on
the unit circle

            float angle
= i * angleStep;


            //"fold"
the sheet around as a
cylinder by placing the
first and last vertex of
each row at the same spot

            if (i ==
numVertexColumns - 1)
                  {


      angle = 0;

                  }

            //position
current vertex

            Vertices[j
* numVertexColumns + i] =
new Vector3(radius *
(float)Math.Cos(angle), j
* heightStep, radius *
(float)Math.Sin(angle));


            //create
the tris

            if (j == 0
|| i >= numVertexColumns
- 1)

            {


      //nothing to do on
the first and last
"floor" on the tris,
capping is done below


      //also nothing to
do on the last column of
vertices


      continue;

            }

            else

            {


      //create 2 tris
below each vertex

                  //6
seems like a magic
number. For every vertex
we draw 2 tris in this
for-loop, therefore we
need 2*3=6 indices in the
Tris array


      //offset the base
by the number of slots we
need for the bottom cap
tris. Those will be
populated once we draw
the cap

                  int
baseIndex = numCapTris *
3 + (j - 1) *
radialSegments * 6 + i *
6;


      //1st tri - below
and in front


      Tris[baseIndex +
0] = j * numVertexColumns
+ i;


      Tris[baseIndex +
1] = j * numVertexColumns
+ i + 1;


      Tris[baseIndex +
2] = (j - 1) *
numVertexColumns + i;


      //2nd tri - the
one it doesn't touch


      Tris[baseIndex +
3] = (j - 1) *
numVertexColumns + i;


      Tris[baseIndex +
4] = j * numVertexColumns
+ i + 1;


      Tris[baseIndex +
5] = (j - 1) *
numVertexColumns + i + 1;

            }

      }
                  }


      //draw caps
                  bool
leftSided = true;
                  int
leftIndex = 0;
```

```
                    int
rightIndex = 0;
                    int
middleIndex = 0;
                    int
topCapVertexOffset =
numVertices -
numVertexColumns;
                    for
(int i = 0; i <
numCapTris; i++)
                    {

      int
bottomCapBaseIndex = i *
3;

      int
topCapBaseIndex =
(numCapTris +
numSideTris) * 3 + i * 3;


      if (i == 0)

      {

            middleIndex
= 0;

            leftIndex =
1;

            rightIndex
= numVertexColumns - 2;

            leftSided =
true;

      }

      else if
(leftSided)

      {

            middleIndex
= rightIndex;

            rightIndex-
-;

      }

      else

      {

            middleIndex
= leftIndex;


      leftIndex++;

      }

      leftSided =
!leftSided;


      //assign bottom
tris

      Tris[bottomCapBase
Index + 0] = rightIndex;

      Tris[bottomCapBase
Index + 1] = middleIndex;

      Tris[bottomCapBase
Index + 2] = leftIndex;


      //assign top tris

      Tris[topCapBaseInd
ex + 0] =
topCapVertexOffset +
leftIndex;

      Tris[topCapBaseInd
ex + 1] =
topCapVertexOffset +
middleIndex;

      Tris[topCapBaseInd
ex + 2] =
topCapVertexOffset +
rightIndex;
                  }

                  for
(int i = 0; i <
Tris.Length; i += 3)
                  {

      triangles.Add(new
Shapes.MyTriangle(Vertice
s[Tris[i]],
Vertices[Tris[i + 1]],
Vertices[Tris[i + 2]]));
                  }

                  for
(int i = 0; i <
triangles.Count; i++)
                  {

      float _a =
triangles[i].a.X +
triangles[i].b.X +
triangles[i].c.X;

      float _b =
triangles[i].a.Y +
triangles[i].b.Y +
triangles[i].c.Y;

      float _c =
triangles[i].a.Z +
triangles[i].b.Z +
triangles[i].c.Z;

      Vector3 v = new
Vector3(_a / 3, _b / 3,
_c / 3);


      triangles[i].norma
l = (triangles[i].c -
v).Cross(triangles[i].a -
v);


      if
(triangles[i].normal.Y <
0) //or whatever
direction up is


      triangles[i].norma
l = -triangles[i].normal;
                  }

            }
      }
}

using Ara3D;
using System;
using
System.Collections.Generi
c;
using System.Text;

namespace KP3D.Shapes
{
    class IcoSphere :
Scene.Shape
    {

        public
IcoSphere(int level = 3)
        {
            type =
"IcoSphere";
            triangles =
new List<MyTriangle>();

Create(level);
        }

        private struct
TriangleIndices
        {
            public int
v1;
            public int
v2;
            public int
v3;
```

```
            public
TriangleIndices(int v1,
int v2, int v3)
            {
                this.v1 =
v1;
                this.v2 =
v2;
                this.v3 =
v3;
            }
        }

        // return index
of point in the middle of
p1 and p2
        public static int
getMiddlePoint(int p1,
int p2, ref List<Vector3>
vertices, ref
Dictionary<long, int>
cache, float radius)
        {
            // first
check if we have it
already
            bool
firstIsSmaller = p1 < p2;
            long
smallerIndex =
firstIsSmaller ? p1 : p2;
            long
greaterIndex =
firstIsSmaller ? p2 : p1;
            long key =
(smallerIndex << 32) +
greaterIndex;

            int ret;
            if
(cache.TryGetValue(key,
out ret))
            {
                return
ret;
            }

            // not in
cache, calculate it
            Vector3
point1 = vertices[p1];
            Vector3
point2 = vertices[p2];
            Vector3
middle = new Vector3
            (
                (point1.X
+ point2.X) / 2f,
                (point1.Y
+ point2.Y) / 2f,
                (point1.Z
+ point2.Z) / 2f
            );

            // add vertex
makes sure point is on
unit sphere
            int i =
vertices.Count;

vertices.Add(middle.Norma
lize() * radius);

            // store it,
return index

cache.Add(key, i);

            return i;
        }

        public void
Create(int
recursionLevell=3)
        {

            List<Vector3>
vertList = new
List<Vector3>();

Dictionary<long, int>
middlePointIndexCache =
new Dictionary<long,
int>();
            int index =
0;

            int
recursionLevel =
recursionLevell;
            float radius
= 1f;

            // create 12
vertices of a icosahedron
            float t = (1f
+ (float)Math.Sqrt(5f)) /
2f;


vertList.Add(new
Vector3(-1f, t,
0f).Normalize() *
radius);

vertList.Add(new
Vector3(1f, t,
0f).Normalize() *
radius);

vertList.Add(new
Vector3(-1f, -t,
0f).Normalize() *
radius);

vertList.Add(new
Vector3(1f, -t,
0f).Normalize() *
radius);


vertList.Add(new
Vector3(0f, -1f,
t).Normalize() * radius);

vertList.Add(new
Vector3(0f, 1f,
t).Normalize() * radius);

vertList.Add(new
Vector3(0f, -1f, -
t).Normalize() * radius);

vertList.Add(new
Vector3(0f, 1f, -
t).Normalize() * radius);


vertList.Add(new
Vector3(t, 0f, -
1f).Normalize() *
radius);

vertList.Add(new
Vector3(t, 0f,
1f).Normalize() *
radius);

vertList.Add(new
Vector3(-t, 0f, -
1f).Normalize() *
radius);

vertList.Add(new
Vector3(-t, 0f,
1f).Normalize() *
radius);


            // create 20
triangles of the
icosahedron

List<TriangleIndices>
faces = new
List<TriangleIndices>();

            // 5 faces
around point 0
            faces.Add(new
TriangleIndices(0, 11,
5));
            faces.Add(new
TriangleIndices(0, 5,
1));
            faces.Add(new
TriangleIndices(0, 1,
7));
            faces.Add(new
TriangleIndices(0, 7,
10));
```

```
            faces.Add(new
TriangleIndices(0, 10,
11));

            // 5 adjacent
faces
            faces.Add(new
TriangleIndices(1, 5,
9));
            faces.Add(new
TriangleIndices(5, 11,
4));
            faces.Add(new
TriangleIndices(11, 10,
2));
            faces.Add(new
TriangleIndices(10, 7,
6));
            faces.Add(new
TriangleIndices(7, 1,
8));

            // 5 faces
around point 3
            faces.Add(new
TriangleIndices(3, 9,
4));
            faces.Add(new
TriangleIndices(3, 4,
2));
            faces.Add(new
TriangleIndices(3, 2,
6));
            faces.Add(new
TriangleIndices(3, 6,
8));
            faces.Add(new
TriangleIndices(3, 8,
9));

            // 5 adjacent
faces
            faces.Add(new
TriangleIndices(4, 9,
5));
            faces.Add(new
TriangleIndices(2, 4,
11));
            faces.Add(new
TriangleIndices(6, 2,
10));
            faces.Add(new
TriangleIndices(8, 6,
7));
            faces.Add(new
TriangleIndices(9, 8,
1));


            // refine
triangles
            for (int i =
0; i < recursionLevel;
i++)
            {
List<TriangleIndices>
faces2 = new
List<TriangleIndices>();
                foreach
(var tri in faces)
                {
                    //
replace triangle by 4
triangles
                    int a
= getMiddlePoint(tri.v1,
tri.v2, ref vertList, ref
middlePointIndexCache,
radius);
                    int b
= getMiddlePoint(tri.v2,
tri.v3, ref vertList, ref
middlePointIndexCache,
radius);
                    int c
= getMiddlePoint(tri.v3,
tri.v1, ref vertList, ref
middlePointIndexCache,
radius);


faces2.Add(new
TriangleIndices(tri.v1,
a, c));

faces2.Add(new
TriangleIndices(tri.v2,
b, a));

faces2.Add(new
TriangleIndices(tri.v3,
c, b));

faces2.Add(new
TriangleIndices(a, b,
c));
                }
                faces =
faces2;
            }

            var vertices
= vertList.ToArray();

            //List<int>
triList = new
List<int>();
            for (int i =
0; i < faces.Count; i++)
            {

//triList.Add(faces[i].v1
);

//triList.Add(faces[i].v2
);

//triList.Add(faces[i].v3
);


triangles.Add(new
Shapes.MyTriangle(vertice
s[faces[i].v1],
vertices[faces[i].v2],
vertices[faces[i].v3]));

            }
            for (int i =
0; i < triangles.Count;
i++)
            {
                float _a
= triangles[i].a.X +
triangles[i].b.X +
triangles[i].c.X;
                float _b
= triangles[i].a.Y +
triangles[i].b.Y +
triangles[i].c.Y;
                float _c
= triangles[i].a.Z +
triangles[i].b.Z +
triangles[i].c.Z;
                Vector3 v
= new Vector3(_a / 3, _b
/ 3, _c / 3);


triangles[i].normal =
(triangles[i].c -
v).Cross(triangles[i].a -
v);

                //одна
вершина

//triangles[i].normal =
(triangles[i].c -
triangles[i].b).Cross(tri
angles[i].a -
triangles[i].b).Normalize
();
                if
(triangles[i].normal.Y <
0) //or whatever
direction up is

triangles[i].normal = -
triangles[i].normal;
            }


            //var
temp_triangles =
triList.ToArray();
            //var uv =
new
Vector2[vertices.Length];
```

```
            // Vector3[]
normales = new
Vector3[vertList.Count];
            //for (int i
= 0; i < vertList.Count;
i++)

//triangles[i].normal =
vertList[i].Normalize();


            //var normals
= normales;

            //var
new_vertices = new
Vector3[triangles.Length*
3];


            //return
new_vertices;
        }
    }
}

using Ara3D;
using System;
using
System.Collections.Generi
c;
using System.Text;

namespace KP3D.Shapes
{
    class Toroid :
Scene.Shape
    {
        public Toroid(int
nbrRingSegments=64, int
nbrTubeSegments = 32,
float ringRad = 1f, float
tubeRadX = 0.35f, float
tubeRadY = 0.35f, int
tubeZeroPos = 90)
        {
            type =
"Toroid";
            triangles =
new List<MyTriangle>();

ToTriangles(nbrRingSegmen
ts, nbrTubeSegments,
ringRad, tubeRadX,
tubeRadY, tubeZeroPos);
        }

        public void
ToTriangles(int
nbrRingSegments = 64, int
nbrTubeSegments = 32,
float ringRad = 1f, float
tubeRadX = 0.35f, float
tubeRadY = 0.35f, int
tubeZeroPos = 90)
        {
            var vertices
= new List<Vector3>();
            var indices =
new List<int>();

            //int
nbrRingSegments=64;
            //int
nbrTubeSegments=32;
            //int
tubeZeroPos = 90;


            //float
ringRad = 1f;
            //float
tubeRadX = 0.35f;
            //float
tubeRadY = 0.35f;

            int
nbrRingSteps =
nbrRingSegments+1;
            int
nbrTubeSteps =
nbrTubeSegments+1;

            Vector3[,]
coord = new
Vector3[nbrRingSteps,
nbrTubeSteps];
            // Calculate
segment size in radians
            float
ringDeltaAng = (float)(2
* Math.PI /
nbrRingSegments);
            float
tubeDeltaAng = (float)(2
* Math.PI /
nbrTubeSegments);

            // Calculate
the XY coordinates of the
tube in the Z=0 plane
            for (int t =
0; t < nbrTubeSteps; t++)
            {
                float
angle = tubeZeroPos + t *
tubeDeltaAng;

coord[0,t] = new Vector3(

ringRad + tubeRadX *
(float)Math.Cos(angle),

tubeRadY *
(float)Math.Sin(angle),
                        0
                    );
            }
            // Calculate
all the points for all
the other ring segments
            for (int r =
1; r < nbrRingSteps; r++)
            {
                float
angle = r * ringDeltaAng;
                float
sinA =
(float)Math.Sin(angle);
                float
cosA =
(float)Math.Cos(angle);
                for (int
t = 0; t < nbrTubeSteps;
t++)
                {

Vector3 point0 = coord[0,
t];

coord[r, t] = new
Vector3(

point0.X * cosA,

point0.Y,

point0.X * sinA
                        );
                }
            }
            // Transfer
to float array
            Vector3[]
points = new
Vector3[nbrRingSteps *
nbrTubeSteps];
            int id = 0;
            for (int t =
0; t < nbrTubeSteps; t++)
            {
                for (int
r = 0; r < nbrRingSteps;
r++)
                {

points[id++] = new
Vector3((float)coord[r,
t].X, (float)coord[r,
t].Y, (float)coord[r,
t].Z);
                }
            }

            int idx = 0;

            for (int t =
0; t < nbrTubeSegments;
t++)
            {
```

```
                for (int
r = 0; r <
nbrRingSegments; r++)
                {
                    int
idxTL = r + t *
nbrRingSteps;
                    int
idxBL = r + (t + 1) *
nbrRingSteps;
                    int
idxBR = r + 1 + (t + 1) *
nbrRingSteps;
                    int
idxTR = r + 1 + t *
nbrRingSteps;

triangles.Add(new
MyTriangle(

points[idxTL],

points[idxBL],

points[idxTR]

));

triangles.Add(new
MyTriangle(

points[idxBL],

points[idxBR],

points[idxTR]

));


                }
            }

            for (int i =
0; i < triangles.Count;
i++)
            {
                float _a
= triangles[i].a.X +
triangles[i].b.X +
triangles[i].c.X;
                float _b
= triangles[i].a.Y +
triangles[i].b.Y +
triangles[i].c.Y;
                float _c
= triangles[i].a.Z +
triangles[i].b.Z +
triangles[i].c.Z;
                Vector3 v
= new Vector3(_a / 3, _b
/ 3, _c / 3);
triangles[i].normal =
(triangles[i].c -
v).Cross(triangles[i].a -
v);

                if
(triangles[i].normal.Y <
0) //or whatever
direction up is

triangles[i].normal = -
triangles[i].normal;
            }

        }
    }
}

using Ara3D;
using System;
using
System.Collections.Generi
c;
using System.Diagnostics;
using System.Text;

namespace KP3D.Scene
{
    class Camera
    {
        public Vector3
eye;
        public Vector3
center =
Vector3.MinValue;
        public float fov;

        public
Camera(Vector3 eye, float
fov)
        {
            this.eye =
eye;
            this.fov =
fov;
        }

        public float
pitch = 0.01f;
        public float yaw
= 0.01f;
        public void
OnMouseMove(int deltaX,
int deltaY)
        {
            pitch = pitch
+ deltaX * 0.02f;
            yaw = yaw +
deltaY * 0.02f;

            if
(Math.Abs(pitch - 0) <
1e-4)
                pitch =
0.001f *
Math.Sign(deltaX);
            if
(Math.Abs(yaw - 0) < 1e-
4)
                yaw =
0.001f *
Math.Sign(deltaY);
        }
    }
}

using System;
using
System.Collections.Generi
c;
using System.Text;

namespace KP3D.Scene
{
    class GroupShapes
    {
        public
GroupShapes()
        {
            shapes = new
List<Shape>();
        }

        public string
name = "undefined";

        public float dx =
0;
        public float dy =
0;
        public float dz =
0;

        public float
rotation_x = 0;
        public float
rotation_y = 0;
        public float
rotation_z = 0;

        public float
scale_x = 1;
        public float
scale_y = 1;
        public float
scale_z = 1;

        public
List<Shape> shapes;
    }
}

using KP3D.Shapes;
using System;
using
System.Collections.Generi
c;
```

```
using System.Text;

namespace KP3D.Scene
{
    public class Shape
    {
        public string
type;

        public float dx =
0;
        public float dy =
0;
        public float dz =
0;

        public float
rotation_x = 0;
        public float
rotation_y = 0;
        public float
rotation_z = 0;

        public float
scale_x = 10;
        public float
scale_y = 10;
        public float
scale_z = 10;

        public
System.Drawing.Color
main_clr;
        public
System.Drawing.Color
select_clr =
Render.color(204, 204,
255);

        public
List<MyTriangle>
triangles;
    }
}

using System;
using
System.Collections.Generi
c;
using System.IO;
using System.Linq;
using System.Text;
using
System.Threading.Tasks;
using System.Windows;
using
System.Windows.Controls;
using
System.Windows.Data;
using
System.Windows.Documents;
using
System.Windows.Input;
using
System.Windows.Media;
using
System.Windows.Media.Imag
ing;
using
System.Windows.Navigation
;
using
System.Windows.Shapes;
using System.Drawing;
using Ara3D;
using System.Diagnostics;
using
System.Windows.Threading;
using KP3D.Scene;
using
System.Runtime.Serializat
ion.Formatters.Binary;
using Microsoft.Win32;

namespace KP3D
{
    /// <summary>
    /// Interaction logic
for MainWindow.xaml
    /// </summary>
    ///


    public partial class
MainWindow : Window
    {
        float fov =
(float)Math.PI/1.6f;
        Scene.Scene
scene;

        int width = 550;
        int height = 480;

        public
MainWindow()
        {

InitializeComponent();
            Scene.Camera
camera = new
Scene.Camera(new
Vector3(0, 0, 0), fov);
            scene = new
Scene.Scene(camera);


SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));

            var timer =
new DispatcherTimer();
            timer.Tick +=
new
EventHandler(timer_Tick);


timer.Interval = new
TimeSpan(0, 0, 0, 0, 1);


timer.Start();

RefreshComboBox();
        }

        public void
ApplyChanges(bool render
= true) {

            if
(scene.selected_id != -1)
            {
                var shape
=
scene.groups[scene.select
ed_gid].shapes[scene.sele
cted_id];
                shape.dx
= (float)dx_slider.Value;
                shape.dy
= (float)dy_slider.Value;
                shape.dz
= (float)dz_slider.Value;


shape.scale_x =
(float)scx_slider.Value;

shape.scale_y =
(float)scy_slider.Value;

shape.scale_z =
(float)scz_slider.Value;


shape.rotation_x =
(float)rx_slider.Value;

shape.rotation_y =
(float)ry_slider.Value;

shape.rotation_z =
(float)rz_slider.Value;
            }

            if(render)

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));

        }
```

```
        public void
GenerateShapeView(Scene.S
hape shape)
        {
            tbdx.Text =
shape.dx.ToString();
            tbdy.Text =
shape.dy.ToString();
            tbdz.Text =
shape.dz.ToString();


dx_slider.Value =
shape.dx;

dy_slider.Value =
shape.dy;

dz_slider.Value =
shape.dz;

            tbsx.Text =
shape.scale_x.ToString();
            tbsy.Text =
shape.scale_y.ToString();
            tbsz.Text =
shape.scale_z.ToString();


scx_slider.Value =
shape.scale_x;

scy_slider.Value =
shape.scale_y;

scz_slider.Value =
shape.scale_z;

            tbrx.Text =
shape.rotation_x.ToString
();
            tbry.Text =
shape.rotation_y.ToString
();
            tbrz.Text =
shape.rotation_z.ToString
();


rx_slider.Value =
shape.rotation_x;

ry_slider.Value =
shape.rotation_y;

rz_slider.Value =
shape.rotation_z;



cp.SelectedColor =
System.Windows.Media.Colo
r.FromRgb(shape.main_clr.
R, shape.main_clr.G,
shape.main_clr.B);
        }

        public void
RefreshGroupsCB()
        {

group_selecter.Items.Clea
r();

group_selecter.Items.Add(
"-1 - undefined");
            for (int i =
0; i <
scene.groups.Count; i++)
            {

group_selecter.Items.Add(
$"{i} - " +
scene.groups[i].name);
            }

group_selecter.SelectedIn
dex = scene.selected_gid
+ 1;
        }

        public void
RefreshComboBox()
        {
            if
(scene.selected_gid != -
1)
            {

selecter.Items.Clear();

selecter.Items.Add("-1 -
None");
                for (int
i = 0; i <
scene.groups[scene.select
ed_gid].shapes.Count;
i++)
                {

selecter.Items.Add($"{i}
- " +
scene.groups[scene.select
ed_gid].shapes[i].type);
                }

selecter.SelectedIndex =
scene.selected_id + 1;
            }
        }

        public void
AddShape(string
ShapeName)
        {
            Scene.Shape
shape = null;
            if(ShapeName
== "Box")
            {
                shape =
new Shapes.Box();
            }
            else
if(ShapeName==
"IcoSphere")
            {
                shape =
new Shapes.IcoSphere(3);
            }
            else if
(ShapeName == "Conus")
            {
               shape =
new Shapes.Conus();
            }
            else if
(ShapeName == "Cylinder")
            {
                shape =
new Shapes.Cylinder();
            }
            else
if(ShapeName == "Toroid")
            {
                shape =
new Shapes.Toroid();
            }
            if (shape !=
null)
            {
                shape.dx
= 0;
                shape.dy
= 0;
                shape.dz
= 0;


shape.scale_x = 100;

shape.scale_y = 100;

shape.scale_z = 100;


shape.rotation_x = 0;

shape.rotation_y = 0;

shape.rotation_z = 0;


shape.main_clr =
Render.color(255, 255,
255);
                if
(scene.selected_gid != -
1)
                {
```

```
scene.groups[scene.select
ed_gid].shapes.Add(shape)
;

RefreshComboBox();

selecter.SelectedIndex =
scene.groups[scene.select
ed_gid].shapes.Count;
                }
            }

        }


        private void
timer_Tick(object sender,
EventArgs e)
        {

if(is_rotating.IsChecked
?? true)

scene.camera.yaw +=
0.01f;

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }

        private void
SceneView_MouseDown(objec
t sender,
MouseButtonEventArgs e)
        {

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
            int x =
(int)e.GetPosition(SceneV
iew).X;
            int y =
(int)e.GetPosition(SceneV
iew).Y;
            int id =
scene.id_object_in_pixel[
x + y * scene.width];
            int gid =
scene.id_group_in_pixel[x
+ y * scene.width];
            if (id != -1
&& gid != -1)

Debug.WriteLine($"({x},
{y}):
{scene.groups[gid].shapes
[id].type}");

scene.selected_id = id;

scene.selected_gid = gid;

selecter.SelectedIndex =
scene.selected_id + 1;

group_selecter.SelectedIn
dex = scene.selected_gid
+ 1;
            if
(scene.selected_id != -1)

GenerateShapeView(scene.g
roups[scene.selected_gid]
.shapes[scene.selected_id
]);
            if
(scene.selected_gid != -
1)

LoadGroupInfo(scene.group
s[scene.selected_gid]);


SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }

        BitmapImage
BitmapToImageSource(Bitma
p bitmap)
        {
            using
(MemoryStream memory =
new MemoryStream())
            {

bitmap.Save(memory,
System.Drawing.Imaging.Im
ageFormat.Bmp);

memory.Position = 0;

BitmapImage bitmapimage =
new BitmapImage();

bitmapimage.BeginInit();

bitmapimage.StreamSource
= memory;

bitmapimage.CacheOption =
BitmapCacheOption.OnLoad;

bitmapimage.EndInit();

                return
bitmapimage;
            }
        }

        private void
SceneView_MouseWheel(obje
ct sender,
MouseWheelEventArgs e)
        {
            if (e.Delta >
0)

scene.camera.fov += 0.1f;

            else if
(e.Delta < 0)

scene.camera.fov -= 0.1f;

            if
(scene.camera.fov <= 0)

scene.camera.fov = 0.1f;


SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));



        }

        int px = 0;
        int py = 0;
        private void
SceneView_MouseMove(objec
t sender, MouseEventArgs
e)
        {

            if
(e.LeftButton ==
MouseButtonState.Pressed)
            {
                int x =
(int)e.GetPosition(this).
X;
                int y =
(int)e.GetPosition(this).
Y;
                if(px ==
0 && py == 0)
                {
                    px =
x;
                    py =
y;
                }
                int dx =
x - px;
                int dy =
y - py;
```

```
                //dy = 0;

                if
(Math.Abs(dx) > 1 ||
Math.Abs(dy) > 1) {

scene.camera.OnMouseMove(
-dy, -dx);

scene.camera.center =
Vector3.MinValue;

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));

//Debug.WriteLine(scene.c
amera.yaw);

//Debug.WriteLine(scene.c
amera.pitch);

//Debug.WriteLine(scene.c
amera.eye);
                    px =
x;
                    py =
y;

                }
            }
            else if
(e.RightButton ==
MouseButtonState.Pressed)
            {

            }
            else
            {
                px = 0;
                py = 0;
            }
        }

        private void
SceneView_KeyDown(object
sender, KeyEventArgs e)
        {
            float dz = 0;
            float dy = 0;
            float dx = 0;
            float coef =
10;
            if (e.Key ==
Key.W)
                dz = -
1f*coef;
            else if
(e.Key == Key.S)
                dz = 1f *
coef;
            if (e.Key ==
Key.A)
                dx = -1f
* coef;
            else if
(e.Key == Key.D)
                dx = 1f *
coef;
            else if
(e.Key == Key.U)
                dy = 1f *
coef;
            else if
(e.Key == Key.J)
                dy = -1f
* coef;
            else if
(e.Key == Key.Delete)
            {
                if
(scene.selected_id != -1)
                {

scene.groups[scene.select
ed_gid].shapes.RemoveAt(s
cene.selected_id);

scene.selected_id = -1;

RefreshComboBox();
                }
            }

scene.camera.eye = new
Vector3(scene.camera.eye.
X + dx,
scene.camera.eye.Y + dy,
scene.camera.eye.Z+dz);

Debug.WriteLine(scene.cam
era.eye);

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }

        private void
selecter_SelectionChanged
(object sender,
SelectionChangedEventArgs
e)
        {
            if
(selecter.Items.Count >
0)
            {
                var a =
selecter.SelectedItem.ToS
tring();

scene.selected_id =
int.Parse(a.Split("
")[0]);
                if
(scene.selected_id != -1)

GenerateShapeView(scene.g
roups[scene.selected_gid]
.shapes[scene.selected_id
]);
                else
                {

tbdx.Text = "0";

tbdy.Text = "0";

tbdz.Text = "0";


dx_slider.Value = 0;

dy_slider.Value = 0;

dz_slider.Value = 0;


tbsx.Text = "0";

tbsy.Text = "0";

tbsz.Text = "0";


scx_slider.Value = 0;

scy_slider.Value = 0;

scz_slider.Value = 0;


tbrx.Text = "0";

tbry.Text = "0";

tbrz.Text = "0";


rx_slider.Value = 0;

ry_slider.Value = 0;

rz_slider.Value = 0;
                }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
            }
        }
        private void
cp_SelectedColorChanged(o
bject sender,
```

```
RoutedPropertyChangedEven
tArgs<System.Windows.Medi
a.Color?> e)
        {
            if
(scene.selected_id != -1)

scene.groups[scene.select
ed_gid].shapes[scene.sele
cted_id].main_clr =
Render.color(cp.SelectedC
olor.Value.R,
cp.SelectedColor.Value.G,
cp.SelectedColor.Value.B)
;
        }
        private void
is_lines_Click(object
sender, RoutedEventArgs
e)
        {

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }

        private void
dx_slider_ValueChanged(ob
ject sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbdx.Text =
dx_slider.Value.ToString(
);
            if
(scene.selected_id != -1)
            {
                var shape
=
scene.groups[scene.select
ed_gid].shapes[scene.sele
cted_id];
                shape.dx
= (float)dx_slider.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }
        private void
dy_slider_ValueChanged(ob
ject sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbdy.Text =
dy_slider.Value.ToString(
);
            if
(scene.selected_id != -1)
            {
                var shape
=
scene.groups[scene.select
ed_gid].shapes[scene.sele
cted_id];
                shape.dy
= (float)dy_slider.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }
        private void
dz_slider_ValueChanged(ob
ject sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbdz.Text =
dz_slider.Value.ToString(
);
            if
(scene.selected_id != -1)
            {
                var shape
=
scene.groups[scene.select
ed_gid].shapes[scene.sele
cted_id];
                shape.dz
= (float)dz_slider.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }

        private void
scx_slider_ValueChanged(o
bject sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbsx.Text =
scx_slider.Value.ToString
();
            if
(scene.selected_id != -1)
            {
                var shape
=
scene.groups[scene.select
ed_gid].shapes[scene.sele
cted_id];

shape.scale_x =
(float)scx_slider.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }
        private void
scy_slider_ValueChanged(o
bject sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbsy.Text =
scy_slider.Value.ToString
();
            if
(scene.selected_id != -1)
            {
                var shape
=
scene.groups[scene.select
ed_gid].shapes[scene.sele
cted_id];

shape.scale_y =
(float)scy_slider.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }
        private void
scz_slider_ValueChanged(o
bject sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbsz.Text =
scz_slider.Value.ToString
();
            if
(scene.selected_id != -1)
            {
                var shape
=
scene.groups[scene.select
ed_gid].shapes[scene.sele
cted_id];

shape.scale_z =
(float)scz_slider.Value;
            }

SceneView.Source =
```

```
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }

        private void
rx_slider_ValueChanged(ob
ject sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbrx.Text =
rx_slider.Value.ToString(
);
            if
(scene.selected_id != -1)
            {
                var shape
=
scene.groups[scene.select
ed_gid].shapes[scene.sele
cted_id];

shape.rotation_x =
(float)rx_slider.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }
        private void
ry_slider_ValueChanged(ob
ject sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbry.Text =
ry_slider.Value.ToString(
);
            if
(scene.selected_id != -1)
            {
                var shape
=
scene.groups[scene.select
ed_gid].shapes[scene.sele
cted_id];

shape.rotation_y =
(float)ry_slider.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }
        private void
rz_slider_ValueChanged(ob
ject sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbrz.Text =
rz_slider.Value.ToString(
);
            if
(scene.selected_id != -1)
            {
                var shape
=
scene.groups[scene.select
ed_gid].shapes[scene.sele
cted_id];

shape.rotation_z =
(float)rz_slider.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }


        private void
tbdx_TextChanged(object
sender,
TextChangedEventArgs e)
        {
            try
            {
                if
(tbdx.Text != "")

dx_slider.Value =
float.Parse(tbdx.Text);
            }
            catch
(Exception)
            {

            };
}
        private void
tbdy_TextChanged(object
sender,
TextChangedEventArgs e)
        {
            try {
            if (tbdy.Text
!= "")

dy_slider.Value =
float.Parse(tbdy.Text);
            }
            catch
(Exception)
            {

            };
        }
        private void
tbdz_TextChanged(object
sender,
TextChangedEventArgs e)
        {
            try
            {
                if
(tbdz.Text != "")

dz_slider.Value =
float.Parse(tbdz.Text);
            }
            catch
(Exception) {

            };
        }

        private void
tbsx_TextChanged(object
sender,
TextChangedEventArgs e)
        {
            try {
            if (tbsx.Text
!= "")

scx_slider.Value =
float.Parse(tbsx.Text);
            }
            catch
(Exception)
            {

            };
        }
        private void
tbsy_TextChanged(object
sender,
TextChangedEventArgs e)
        {
            try
            {
                if
(tbsy.Text != "")

scy_slider.Value =
float.Parse(tbsy.Text);
            }
            catch
(Exception)
            {

            };
}
        private void
tbsz_TextChanged(object
sender,
TextChangedEventArgs e)
        {
            try {
```

```
            if (tbsz.Text
!= "")

scz_slider.Value =
float.Parse(tbsz.Text);
            }
            catch
(Exception)
            {

            };
        }

        private void
tbrx_TextChanged(object
sender,
TextChangedEventArgs e)
        {
            try {
            if (tbrx.Text
!= "")

rx_slider.Value =
float.Parse(tbrx.Text);
            }
            catch
(Exception)
            {

            };
        }
        private void
tbry_TextChanged(object
sender,
TextChangedEventArgs e)
        {
            try
            {
                if
(tbry.Text != "")

ry_slider.Value =
float.Parse(tbry.Text);
            }
            catch
(Exception)
            {

            };
        }
        private void
tbrz_TextChanged(object
sender,
TextChangedEventArgs e)
        {
            try
            {
                if
(tbrz.Text != "")

rz_slider.Value =
float.Parse(tbrz.Text);
            }
            catch
(Exception)
            {

            };
}



        private void
dx_slider_ValueChanged2(o
bject sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbdx2.Text =
dx_slider2.Value.ToString
();
            if
(scene.selected_gid != -
1)
            {
                var shape
=
scene.groups[scene.select
ed_gid];
                shape.dx
=
(float)dx_slider2.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }
        private void
dy_slider_ValueChanged2(o
bject sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbdy2.Text =
dy_slider2.Value.ToString
();
            if
(scene.selected_gid != -
1)
            {
                var shape
=
scene.groups[scene.select
ed_gid];
                shape.dy
=
(float)dy_slider2.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }

        private void
dz_slider_ValueChanged2(o
bject sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbdz2.Text =
dz_slider2.Value.ToString
();
            if
(scene.selected_gid != -
1)
            {
                var shape
=
scene.groups[scene.select
ed_gid];
                shape.dz
=
(float)dz_slider2.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }

        private void
scx_slider_ValueChanged2(
object sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbsx2.Text =
scx_slider2.Value.ToStrin
g();
            if
(scene.selected_gid != -
1)
            {
                var shape
=
scene.groups[scene.select
ed_gid];

shape.scale_x =
(float)scx_slider2.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }
        private void
scy_slider_ValueChanged2(
object sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
```

```
            tbsy2.Text =
scy_slider2.Value.ToStrin
g();
            if
(scene.selected_gid != -
1)
            {
                var shape
=
scene.groups[scene.select
ed_gid];

shape.scale_y =
(float)scy_slider2.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }
        private void
scz_slider_ValueChanged2(
object sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbsz2.Text =
scz_slider2.Value.ToStrin
g();
            if
(scene.selected_gid != -
1)
            {
                var shape
=
scene.groups[scene.select
ed_gid];

shape.scale_z =
(float)scz_slider2.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }

        private void
rx_slider_ValueChanged2(o
bject sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbrx2.Text =
rx_slider2.Value.ToString
();
            if
(scene.selected_gid != -
1)
            {
                var shape
=
scene.groups[scene.select
ed_gid];

shape.rotation_x =
(float)rx_slider2.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }
        private void
ry_slider_ValueChanged2(o
bject sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbry2.Text =
ry_slider2.Value.ToString
();
            if
(scene.selected_gid != -
1)
            {
                var shape
=
scene.groups[scene.select
ed_gid];

shape.rotation_y =
(float)ry_slider2.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }
        private void
rz_slider_ValueChanged2(o
bject sender,
RoutedPropertyChangedEven
tArgs<double> e)
        {
            tbrz2.Text =
rz_slider2.Value.ToString
();
            if
(scene.selected_gid != -
1)
            {
                var shape
=
scene.groups[scene.select
ed_gid];

shape.rotation_z =
(float)rz_slider2.Value;
            }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }


        private void
tbdx_TextChanged2(object
sender,
TextChangedEventArgs e)
        {
            try
            {
                if
(tbdx2.Text != "")

dx_slider2.Value =
float.Parse(tbdx2.Text);
            }
            catch
(Exception)
            {

            };
        }
        private void
tbdy_TextChanged2(object
sender,
TextChangedEventArgs e)
        {
            try
            {
                if
(tbdy2.Text != "")

dy_slider2.Value =
float.Parse(tbdy2.Text);
            }
            catch
(Exception)
            {

            };
        }
        private void
tbdz_TextChanged2(object
sender,
TextChangedEventArgs e)
        {
            try
            {
                if
(tbdz2.Text != "")

dz_slider2.Value =
float.Parse(tbdz2.Text);
            }
            catch
(Exception)
            {
```

```
            };
        }

        private void
tbsx_TextChanged2(object
sender,
TextChangedEventArgs e)
        {
            try
            {
                if
(tbsx2.Text != "")

scx_slider2.Value =
float.Parse(tbsx2.Text);
            }
            catch
(Exception)
            {

            };
        }
        private void
tbsy_TextChanged2(object
sender,
TextChangedEventArgs e)
        {
            try
            {
                if
(tbsy2.Text != "")

scy_slider2.Value =
float.Parse(tbsy2.Text);
            }
            catch
(Exception)
            {

            };
        }
        private void
tbsz_TextChanged2(object
sender,
TextChangedEventArgs e)
        {
            try
            {
                if
(tbsz2.Text != "")

scz_slider2.Value =
float.Parse(tbsz2.Text);
            }
            catch
(Exception)
            {

            };
        }

        private void
tbrx_TextChanged2(object
sender,
TextChangedEventArgs e)
        {
            try
            {
                if
(tbrx2.Text != "")

rx_slider2.Value =
float.Parse(tbrx2.Text);
            }
            catch
(Exception)
            {

            };
        }
        private void
tbry_TextChanged2(object
sender,
TextChangedEventArgs e)
        {
            try
            {
                if
(tbry2.Text != "")

ry_slider2.Value =
float.Parse(tbry2.Text);
            }
            catch
(Exception)
            {

            };
        }
        private void
tbrz_TextChanged2(object
sender,
TextChangedEventArgs e)
        {
            try
            {
                if
(tbrz2.Text != "")

rz_slider2.Value =
float.Parse(tbrz2.Text);
            }
            catch
(Exception)
            {

            };
        }

        private void
gname_TextChanged(object
sender,
TextChangedEventArgs e)
        {
            if (scene !=
null &&
scene.selected_gid != -1)
            {

scene.groups[scene.select
ed_gid].name =
gname.Text;

RefreshGroupsCB();
            }
        }

        private void
Button_Click(object
sender, RoutedEventArgs
e)
        {

AddShape("IcoSphere");
        }

        private void
Button_Click_1(object
sender, RoutedEventArgs
e)
        {

AddShape("Box");
        }

        private void
Button_Click_2(object
sender, RoutedEventArgs
e)
        {

AddShape("Toroid");
        }

        private void
Button_Click_3(object
sender, RoutedEventArgs
e)
        {

AddShape("Cylinder");
        }

        private void
Button_Click_4(object
sender, RoutedEventArgs
e)
        {

AddShape("Conus");
        }

        private void
Save_Click(object sender,
RoutedEventArgs e)
        {
            var sfd = new
SaveFileDialog();
            sfd.Filter =
"text|*.txt";
```

```
            sfd.Title =
"Save an scene";

sfd.ShowDialog();
            if
(sfd.FileName != "")
            {
                using
(StreamWriter fs = new
StreamWriter(sfd.FileName
))
                {

fs.WriteLine(scene.camera
.eye.X + " " +
scene.camera.eye.Y + " "
+ scene.camera.eye.Z);

fs.WriteLine(scene.camera
.fov);

fs.WriteLine(scene.camera
.pitch);

fs.WriteLine(scene.camera
.yaw);
                    for
(int g = 0; g <
scene.groups.Count; g++)
                    {

fs.WriteLine("__group__")
;

fs.WriteLine(scene.groups
[g].name);


fs.WriteLine(scene.groups
[g].dx);

fs.WriteLine(scene.groups
[g].dy);

fs.WriteLine(scene.groups
[g].dz);


fs.WriteLine(scene.groups
[g].scale_x);

fs.WriteLine(scene.groups
[g].scale_y);

fs.WriteLine(scene.groups
[g].scale_z);


fs.WriteLine(scene.groups
[g].rotation_x);

fs.WriteLine(scene.groups
[g].rotation_y);

fs.WriteLine(scene.groups
[g].rotation_z);


for (int i = 0; i <
scene.groups[g].shapes.Co
unt; i++)
                            {

fs.WriteLine(scene.groups
[g].shapes[i].type);


fs.WriteLine(scene.groups
[g].shapes[i].dx);

fs.WriteLine(scene.groups
[g].shapes[i].dy);

fs.WriteLine(scene.groups
[g].shapes[i].dz);


fs.WriteLine(scene.groups
[g].shapes[i].scale_x);

fs.WriteLine(scene.groups
[g].shapes[i].scale_y);

fs.WriteLine(scene.groups
[g].shapes[i].scale_z);


fs.WriteLine(scene.groups
[g].shapes[i].rotation_x)
;

fs.WriteLine(scene.groups
[g].shapes[i].rotation_y)
;

fs.WriteLine(scene.groups
[g].shapes[i].rotation_z)
;

fs.WriteLine(

scene.groups[g].shapes[i]
.main_clr.R

+ " "

+
scene.groups[g].shapes[i]
.main_clr.G

+ " "

+
scene.groups[g].shapes[i]
.main_clr.B

);
                            }
                    }
                }
            }

        private void
Load_Click(object sender,
RoutedEventArgs e)
        {
            var lfd = new
OpenFileDialog();
            lfd.Filter =
"text|*.txt";
            lfd.Title =
"Load an scene";

lfd.ShowDialog();
            if
(lfd.FileName != "")
            {

scene.groups.Clear();
                using
(StreamReader fs = new
StreamReader(lfd.FileName
))
                {

string line;

                    var
eye =
fs.ReadLine().Split(" ");

scene.camera.eye = new
Vector3(float.Parse(eye[0
]), float.Parse(eye[1]),
float.Parse(eye[2]));

scene.camera.fov =
float.Parse(fs.ReadLine()
);

scene.camera.pitch =
float.Parse(fs.ReadLine()
);

scene.camera.yaw =
float.Parse(fs.ReadLine()
);

                    while
((line = fs.ReadLine())
!= null)
                    {

if (line != "")
                        {

if (line == "__group__")

{
```

```
scene.groups.Add(new
GroupShapes());

scene.groups[scene.groups
.Count - 1].name =
fs.ReadLine();


scene.groups[scene.groups
.Count - 1].dx =
float.Parse(fs.ReadLine()
);

scene.groups[scene.groups
.Count - 1].dy =
float.Parse(fs.ReadLine()
);

scene.groups[scene.groups
.Count - 1].dz =
float.Parse(fs.ReadLine()
);


scene.groups[scene.groups
.Count - 1].scale_x =
float.Parse(fs.ReadLine()
);

scene.groups[scene.groups
.Count - 1].scale_y =
float.Parse(fs.ReadLine()
);

scene.groups[scene.groups
.Count - 1].scale_z =
float.Parse(fs.ReadLine()
);


scene.groups[scene.groups
.Count - 1].rotation_x =
float.Parse(fs.ReadLine()
);

scene.groups[scene.groups
.Count - 1].rotation_y =
float.Parse(fs.ReadLine()
);

scene.groups[scene.groups
.Count - 1].rotation_z =
float.Parse(fs.ReadLine()
);

continue;

}

Scene.Shape shape = new
Scene.Shape();

if (line == "Box")
{

shape = new Shapes.Box();

}

else if (line ==
"IcoSphere")

{

shape = new
Shapes.IcoSphere();

}

else if (line ==
"Toroid")

{

shape = new
Shapes.Toroid();

}

else if (line == "Conus")

{

shape = new
Shapes.Conus();

}

else if (line ==
"Cylinder")

{

shape = new
Shapes.Cylinder();

}


shape.dx =
float.Parse(fs.ReadLine()
);

shape.dy =
float.Parse(fs.ReadLine()
);

shape.dz =
float.Parse(fs.ReadLine()
);


shape.scale_x =
float.Parse(fs.ReadLine()
);

shape.scale_y =
float.Parse(fs.ReadLine()
);

shape.scale_z =
float.Parse(fs.ReadLine()
);


shape.rotation_x =
float.Parse(fs.ReadLine()
);

shape.rotation_y =
float.Parse(fs.ReadLine()
);

shape.rotation_z =
float.Parse(fs.ReadLine()
);


var clr =
fs.ReadLine().Split(" ");

shape.main_clr =
Render.color(int.Parse(cl
r[0]), int.Parse(clr[1]),
int.Parse(clr[2]));

scene.groups[scene.groups
.Count -
1].shapes.Add(shape);
                        }
                    }

RefreshComboBox();
                }
            }
        }
        private void
Button_Click_5(object
sender, RoutedEventArgs
e)
        {
            float feye_x
=
float.Parse(eye_x.Text);
            float feye_y
=
float.Parse(eye_y.Text);
            float feye_z
=
float.Parse(eye_z.Text);

scene.camera.eye = new
Vector3(feye_x, feye_y,
feye_z);

            float fdir_x
=
float.Parse(dir_x.Text);
```

```
            float fdir_y
=
float.Parse(dir_y.Text);
            float fdir_z
=
float.Parse(dir_z.Text);


scene.camera.center = new
Vector3(fdir_x, fdir_y,
fdir_z);

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
        }

        //add group
        private void
Button_Click_6(object
sender, RoutedEventArgs
e)
        {

scene.groups.Add(new
GroupShapes());

RefreshGroupsCB();

group_selecter.SelectedIn
dex = scene.groups.Count;

scene.selected_gid =
scene.groups.Count - 1;
        }

        void
LoadGroupInfo(GroupShapes
shape)
        {
            gname.Text =
shape.name.ToString();
            tbdx2.Text =
shape.dx.ToString();
            tbdy2.Text =
shape.dy.ToString();
            tbdz2.Text =
shape.dz.ToString();


dx_slider2.Value =
shape.dx;

dy_slider2.Value =
shape.dy;

dz_slider2.Value =
shape.dz;

            tbsx2.Text =
shape.scale_x.ToString();
            tbsy2.Text =
shape.scale_y.ToString();
            tbsz2.Text =
shape.scale_z.ToString();


scx_slider2.Value =
shape.scale_x;

scy_slider2.Value =
shape.scale_y;

scz_slider2.Value =
shape.scale_z;

            tbrx2.Text =
shape.rotation_x.ToString
();
            tbry2.Text =
shape.rotation_y.ToString
();
            tbrz2.Text =
shape.rotation_z.ToString
();


rx_slider2.Value =
shape.rotation_x;

ry_slider2.Value =
shape.rotation_y;

rz_slider2.Value =
shape.rotation_z;
        }

        private void
group_selecter_SelectionC
hanged(object sender,
SelectionChangedEventArgs
e)
        {
            if
(group_selecter.Items.Cou
nt > 0)
            {
                var a =
group_selecter.SelectedIt
em.ToString();

scene.selected_gid =
int.Parse(a.Split("
")[0]);

                if
(scene.selected_gid != -
1)
                {

LoadGroupInfo(scene.group
s[scene.selected_gid]);

RefreshComboBox();
                }
                else
                {

gname.Text = "undefined";

tbdx2.Text = "0";

tbdy2.Text = "0";

tbdz2.Text = "0";


dx_slider2.Value = 0;

dy_slider2.Value = 0;

dz_slider2.Value = 0;


tbsx2.Text = "0";

tbsy2.Text = "0";

tbsz2.Text = "0";


scx_slider2.Value = 0;

scy_slider2.Value = 0;

scz_slider2.Value = 0;


tbrx2.Text = "0";

tbry2.Text = "0";

tbrz2.Text = "0";


rx_slider2.Value = 0;

ry_slider2.Value = 0;

rz_slider2.Value = 0;
                }

SceneView.Source =
BitmapToImageSource(scene
.render(width, height,
is_lines.IsChecked ??
false));
            }
        }

        private void
Button_Click_7(object
sender, RoutedEventArgs
e)
        {
            if
(scene.selected_gid != -
1)
            {
```

```
scene.groups.RemoveAt(sce
ne.selected_gid);

scene.selected_gid = -1;

scene.selected_id = -1;

RefreshComboBox();

RefreshGroupsCB();
            }
        }

        private void
SaveGroup_Click(object
sender, RoutedEventArgs
e)
        {
            if
(scene.selected_gid != -
1)
            {
                var sfd =
new SaveFileDialog();

sfd.Filter =
"text|*.txt";
                sfd.Title
= "Save an object";

sfd.ShowDialog();
                if
(sfd.FileName != "")
                {
                    using
(StreamWriter fs = new
StreamWriter(sfd.FileName
))
                    {

fs.WriteLine(scene.groups
[scene.selected_gid].name
);


for (int i = 0; i <
scene.groups[scene.select
ed_gid].shapes.Count;
i++)
                        {

fs.WriteLine(scene.groups
[scene.selected_gid].shap
es[i].type);


fs.WriteLine(scene.groups
[scene.selected_gid].shap
es[i].dx);

fs.WriteLine(scene.groups
[scene.selected_gid].shap
es[i].dy);

fs.WriteLine(scene.groups
[scene.selected_gid].shap
es[i].dz);


fs.WriteLine(scene.groups
[scene.selected_gid].shap
es[i].scale_x);

fs.WriteLine(scene.groups
[scene.selected_gid].shap
es[i].scale_y);

fs.WriteLine(scene.groups
[scene.selected_gid].shap
es[i].scale_z);


fs.WriteLine(scene.groups
[scene.selected_gid].shap
es[i].rotation_x);

fs.WriteLine(scene.groups
[scene.selected_gid].shap
es[i].rotation_y);

fs.WriteLine(scene.groups
[scene.selected_gid].shap
es[i].rotation_z);

fs.WriteLine(

scene.groups[scene.select
ed_gid].shapes[i].main_cl
r.R

+ " "

+
scene.groups[scene.select
ed_gid].shapes[i].main_cl
r.G

+ " "

+
scene.groups[scene.select
ed_gid].shapes[i].main_cl
r.B

);
                        }
                    }
                }
            }
        }

        private void
LoadGroup_Click(object
sender, RoutedEventArgs
e)
        {
            var lfd = new
OpenFileDialog();
            lfd.Filter =
"text|*.txt";
            lfd.Title =
"Load an scene";

lfd.ShowDialog();
            if
(lfd.FileName != "")
            {

scene.groups.Add(new
GroupShapes());

scene.selected_gid =
scene.groups.Count - 1;
                using
(StreamReader fs = new
StreamReader(lfd.FileName
))
                {

string line;

scene.groups[scene.select
ed_gid].name =
fs.ReadLine();

                    while
((line = fs.ReadLine())
!= null)
                    {

if (line != "")
                        {

Scene.Shape shape = new
Scene.Shape();

if (line == "Box")

{

shape = new Shapes.Box();

}

else if (line ==
"IcoSphere")

{

shape = new
Shapes.IcoSphere();

}

else if (line ==
"Toroid")

{
```

```
shape = new
Shapes.Toroid();

}

else if (line == "Conus")

{

shape = new
Shapes.Conus();

}

else if (line ==
"Cylinder")

{

shape = new
Shapes.Cylinder();

}


shape.dx =
float.Parse(fs.ReadLine()
);

shape.dy =
float.Parse(fs.ReadLine()
);

shape.dz =
float.Parse(fs.ReadLine()
);


shape.scale_x =
float.Parse(fs.ReadLine()
);

shape.scale_y =
float.Parse(fs.ReadLine()
);

shape.scale_z =
float.Parse(fs.ReadLine()
);


shape.rotation_x =
float.Parse(fs.ReadLine()
);

shape.rotation_y =
float.Parse(fs.ReadLine()
);

shape.rotation_z =
float.Parse(fs.ReadLine()
);


var clr =
fs.ReadLine().Split(" ");

shape.main_clr =
Render.color(int.Parse(cl
r[0]), int.Parse(clr[1]),
int.Parse(clr[2]));

scene.groups[scene.select
ed_gid].shapes.Add(shape)
;
                        }
                    }
                }

RefreshGroupsCB();

group_selecter.SelectedIn
dex = scene.groups.Count;
            }
        }


    }
}
```

ПРИЛОЖЕНИЕ Д. СПРАВКА АНТИПЛАГИАТА

АНТИПЛАГИАТ
ТВОРИТЕ СОБСТВЕННЫМ УМОМ

Отчет о проверке на заимствования №1

Автор: dimapetrenko2000@gmail.com / ID: 7288244
Проверяющий: (dimapetrenko2000@gmail.com / ID: 7288244)

Отчет предоставлен сервисом «Антиплагиат»- http://users.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 8
Начало загрузки: 28.05.2020 22:21:12
Длительность загрузки: 00:00:03
Имя исходного файла:
пз_графика_Петренко_д.а..pdf
Название документа:
пз_графика_Петренко_д.а.
Размер текста: 1 кБ
Символов в тексте: 94638
Слов в тексте: 12472
Число предложений: 1675

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 28.05.2020 22:21:15
Длительность проверки: 00:00:03
Комментарии: не указано
Модули поиска: Модуль поиска Интернет

ЗАИМСТВОВАНИЯ	САМОЦИТИРОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
9,87%	0%	0%	90,13%

Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Самоцитирования — доля фрагментов текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты; общеупотребительные выражения; фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Заимствования, самоцитирования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.