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

1 РАЗРАБОТКА ПОЛИГОНАЛЬНОЙ МОДЕЛИ ОБЪЕКТА 6

1.1 Составляющие элементы объекта 6

1.2 Триангуляция поверхности объекта 7

2 ОПИСАНИЕ ВЫБРАННЫХ МЕТОДОВ И АЛГОРИТМОВ ВИЗУАЛИЗАЦИИ 9

2.1 Алгоритмы аффинных преобразований 9

2.2 Алгоритм удаления скрытых линий 9

2.3 Алгоритмы проекционных преобразований 10

2.3.1 Алгоритм параллельного проецирования 11

2.3.2 Алгоритм центрального проецирования 11

3 РАЗРАБОТКА СТРУКТУР ДАННЫХ ДЛЯ ХРАНЕНИЯ И ОПИСАНИЯ ОБЪЕКТА 13

3.1 Описание структур данных 13

3.2 UML-диаграммы классов 14

4 РЕАЛИЗАЦИЯ ПРЕОБРАЗОВАНИЙ НАД ОБЪЕКТОМ 16

4.1 Реализация масштабирования 16

4.2 Реализация поворота 17

4.3 Реализация перемещения объекта 18

4.4 Реализация изменения параметров 19

4.5 Реализация перспективной проекции 20

4.6 Реализация сохранения и открытия сцены 21

5 ТЕСТИРОВАНИЕ ПРОГРАММЫ 25

ВЫВОДЫ 29

ПЕРЕЧЕНЬ ССЫЛОК 30

Приложение А. Экранные формы 31

Приложение Б. Техническое задание 33

Приложение В. Руководство пользователя 37

Приложение Г. Листинг программы 39

Приложение Д. Справка антиплагиата 69

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#.

# РАЗРАБОТКА ПОЛИГОНАЛЬНОЙ МОДЕЛИ ОБЪЕКТА

## 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.

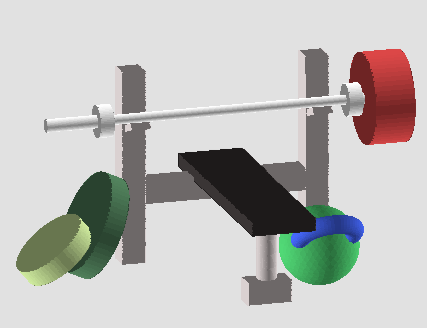


Рисунок 1.1 – Трехмерное изображение проектируемого объекта

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.

## 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.

# ОПИСАНИЕ ВЫБРАННЫХ МЕТОДОВ И АЛГОРИТМОВ ВИЗУАЛИЗАЦИИ

## 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.

## 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*(*х,у*) со значением *Zбуфер*(*х,у*), хранящимся в *z*-буфере в этой же позиции;
5. Если *z*(*х,у*) > *Zбуфер* (*х,у*), то записать атрибут этого многоугольника (интенсивность, цвет и т. п.) в буфер кадра и заменить *Zбуфер*(*х,у*) на *z*(*х,у*). В противном случае никаких действий не производить.

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

## 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.

## 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 4х4.

## 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.



Рисунок 2.1 – Ортогональные координаты точки зрения

Так же в произвольном видовом преобразовании было использовано перспективное проецирование.

# РАЗРАБОТКА СТРУКТУР ДАННЫХ ДЛЯ ХРАНЕНИЯ И ОПИСАНИЯ ОБЪЕКТА

## 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.

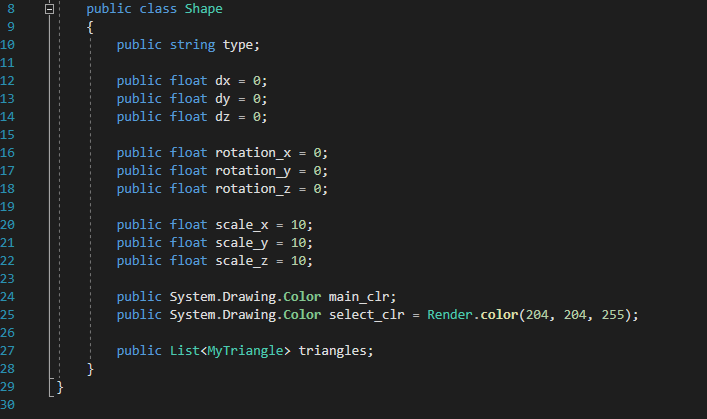


Рисунок 3.1 – Структура класса Shape

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

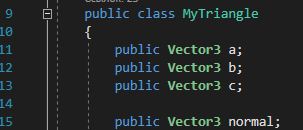


Рисунок 3.2 – Структура класса MyTriangle

## 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.

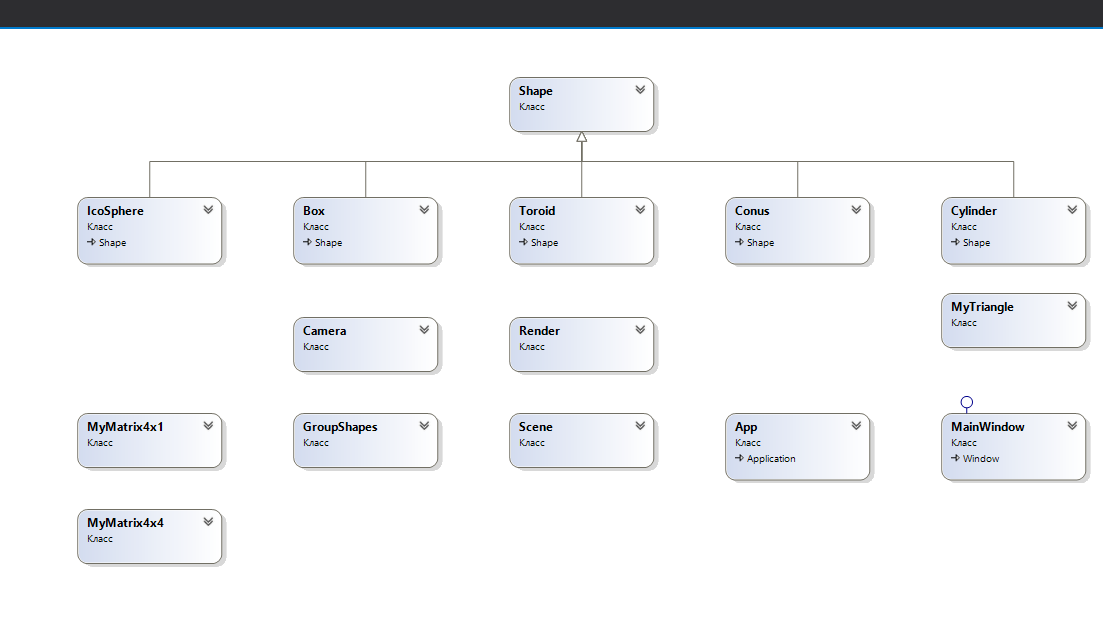


Рисунок 3.3 – Диаграмма отношений классов

# РЕАЛИЗАЦИЯ ПРЕОБРАЗОВАНИЙ НАД ОБЪЕКТОМ

## 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.



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.

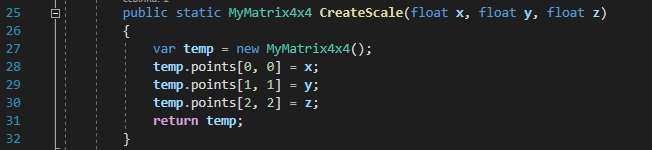


Рисунок 4.2 – Алгоритм масштабирования

## 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



Поворот вокруг оси X



Поворот вокруг оси Y



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.

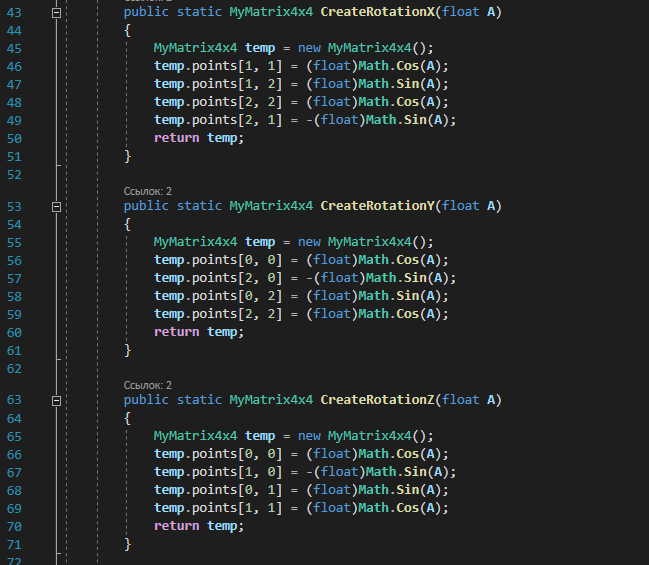


Рисунок 4.4 – Алгоритм поворота объекта

## 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.



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.

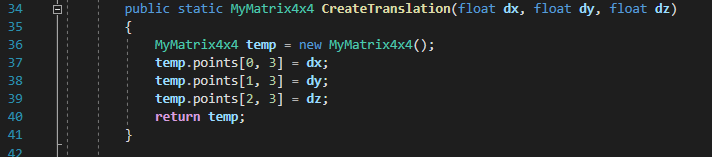


Рисунок 4.6 – Алгоритм переноса объекта

* 1. 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.

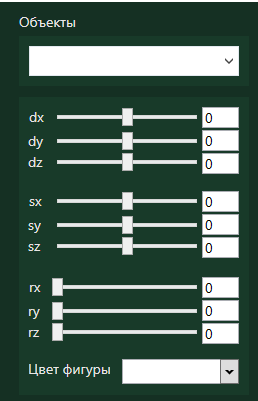


Рисунок 4.7 – Форма ввода параметров объекта

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.
   1. 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.

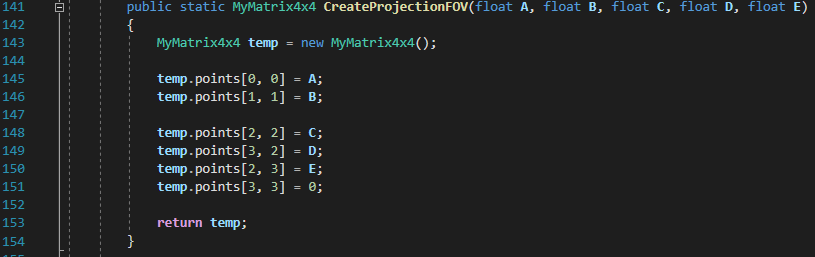


Рисунок 4.8 – Формирование матрицы перспективной проекции

## 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 матрциы 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.

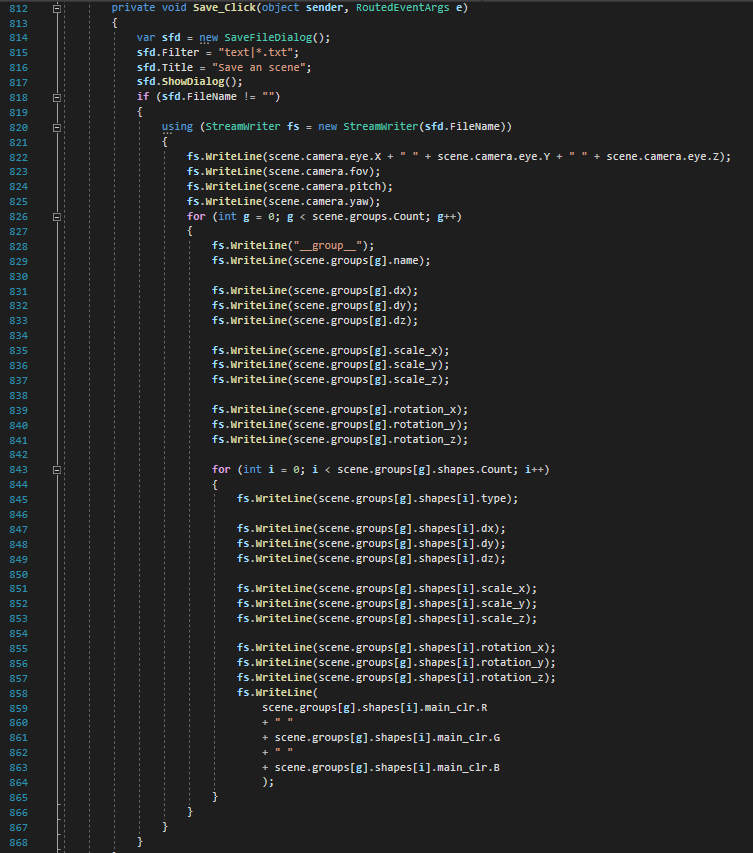


Рисунок 4.10 – Алгоритм записи сцены в файл

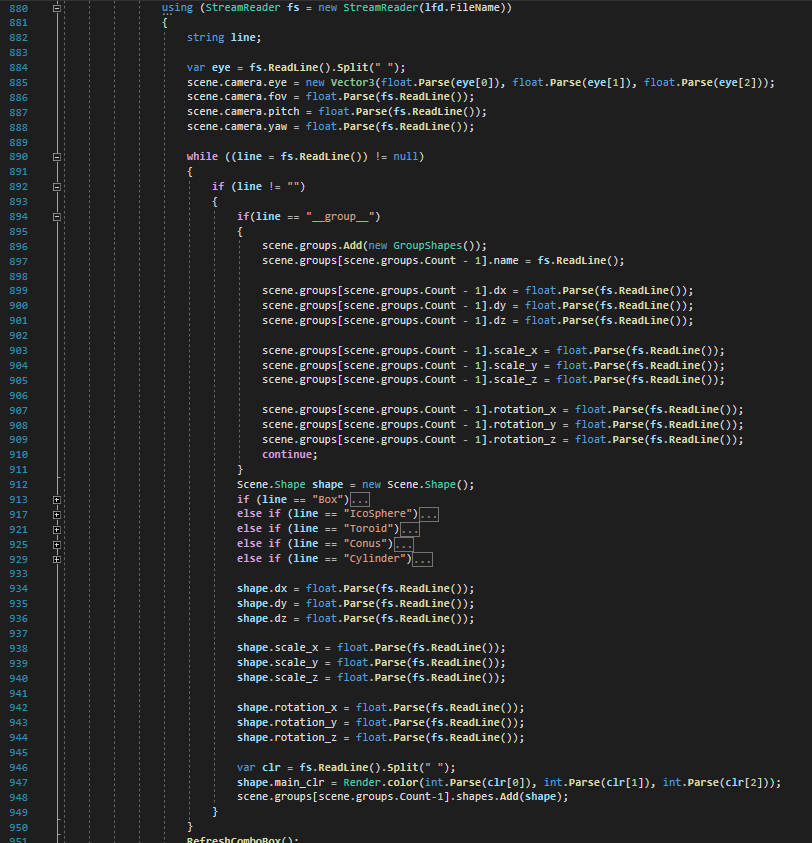


Рисунок 4.11 – Алгоритм чтения сцены из фалйа

# 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

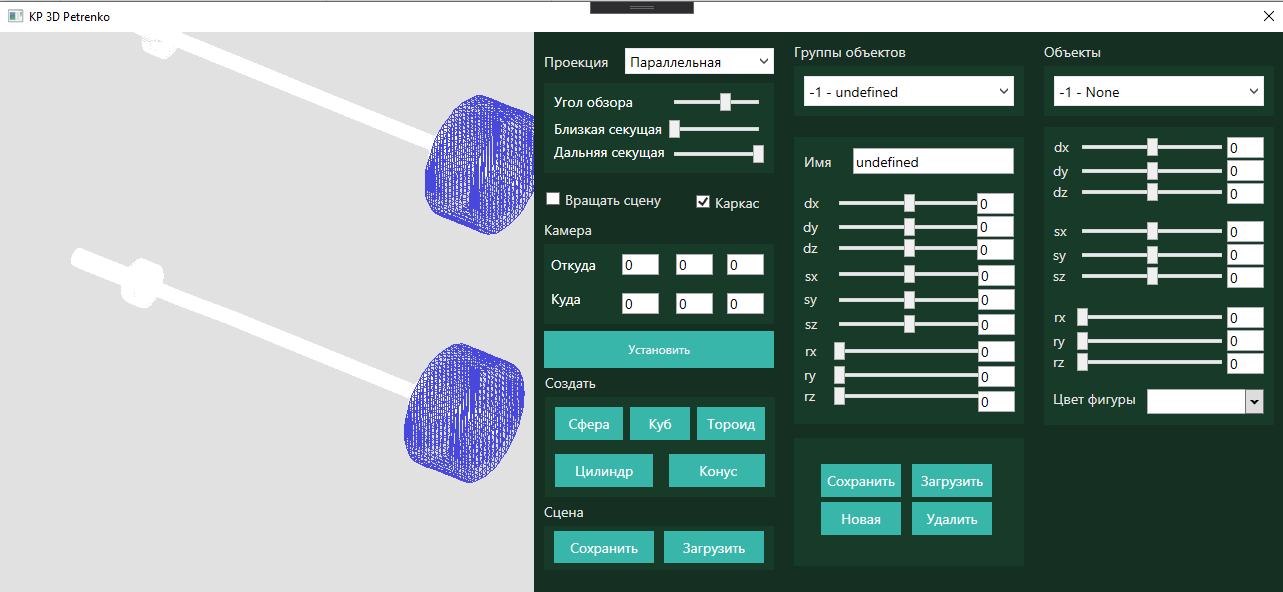


Рисунок A.1

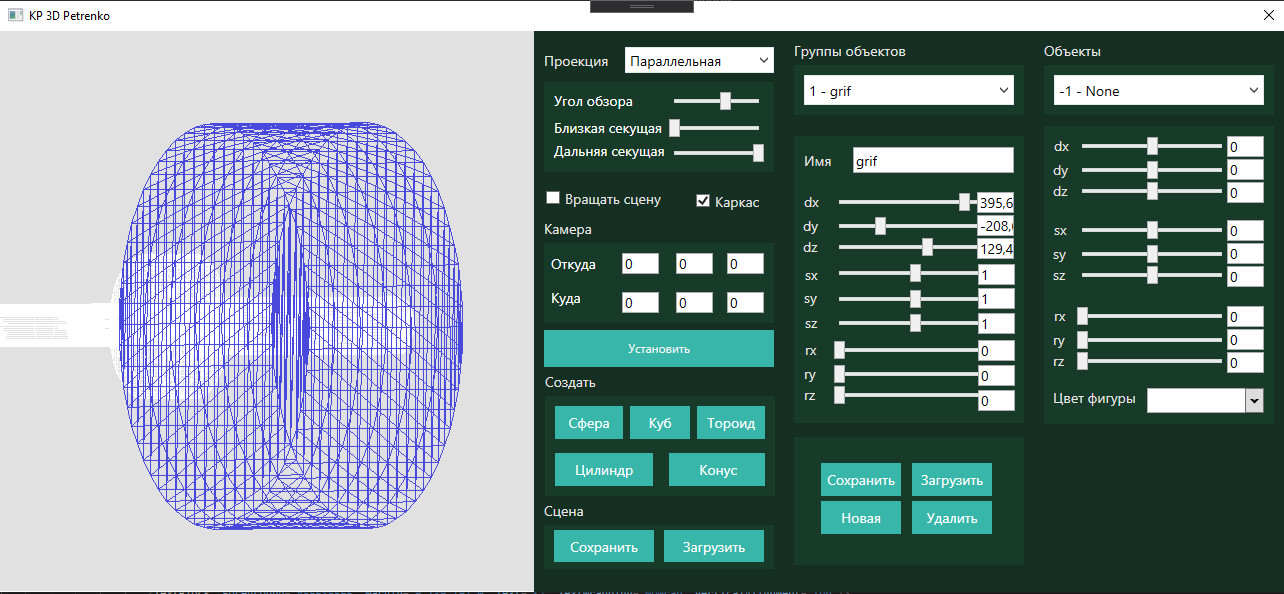


Рисунок A.2

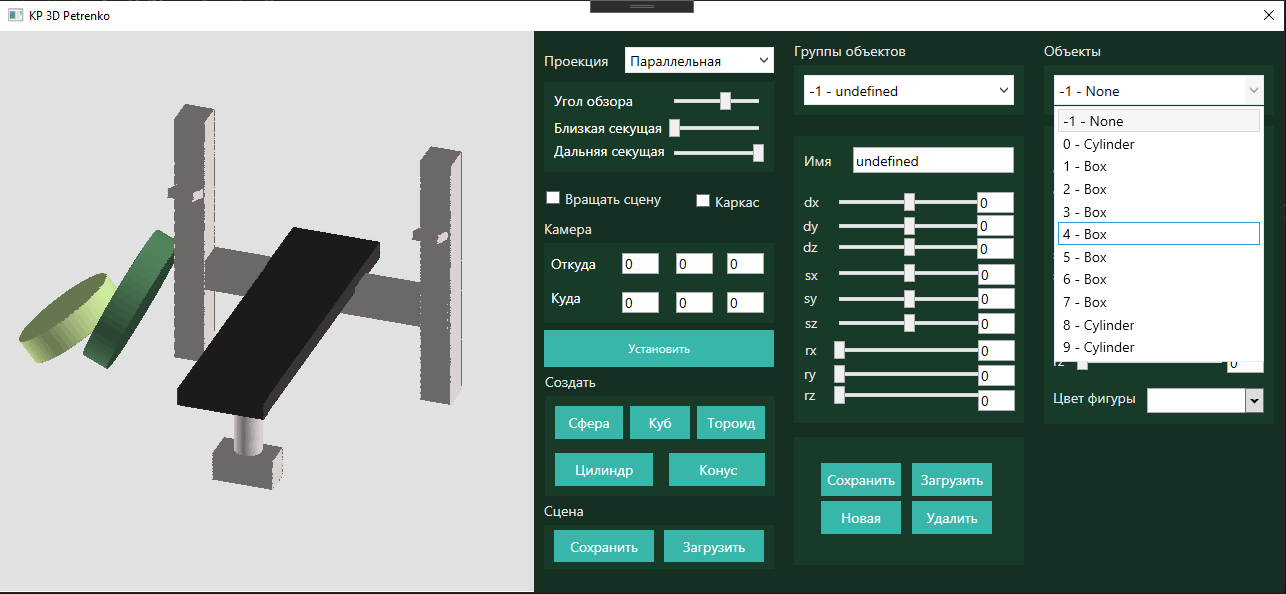


Рисунок A.3

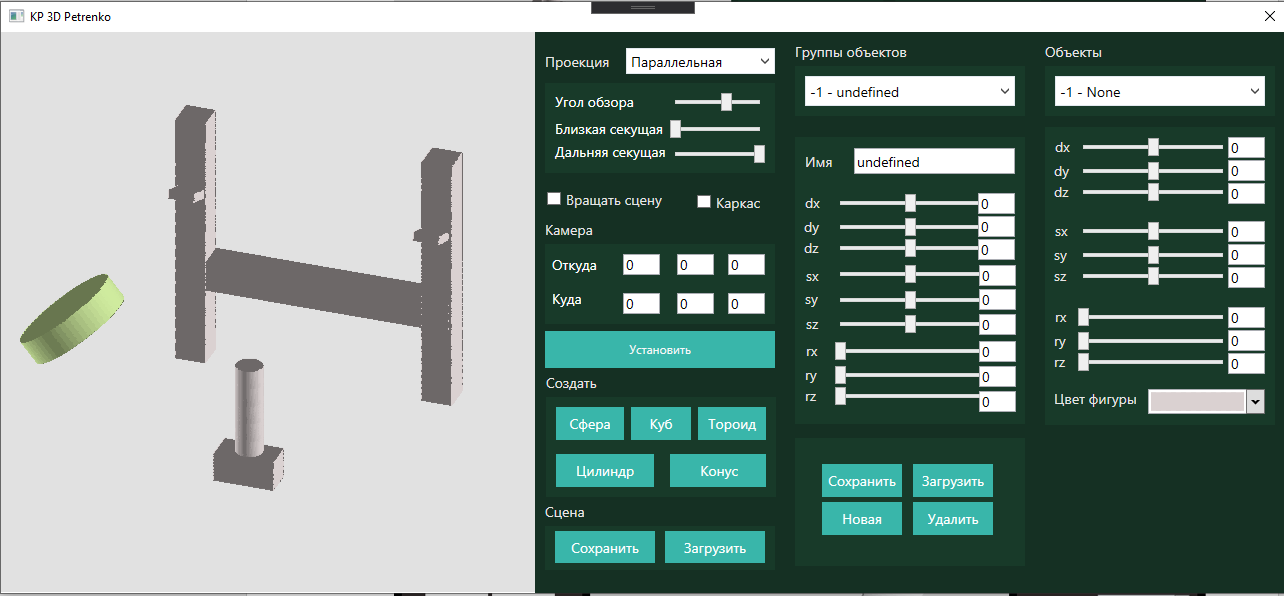


Рисунок A.4

# 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.

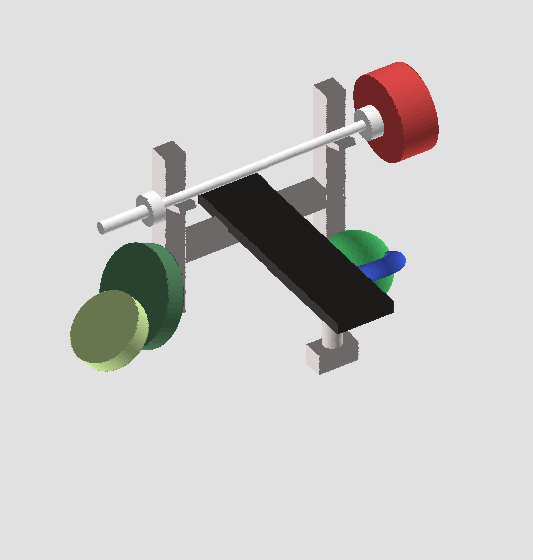


Рисунок А.1- Тренажер

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. Читабельность базы данных:

Файл сцены должен содержать данные модели в текстовом виде.

* 1. Возможность работы с несколькими объектами:

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

1. Обеспечить редактирование и параметризацию объектов:

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

1. Обеспечить центральное и параллельное проецирование:

Возможность переключения с одного вида проецирования на другой

1. 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.

1. Удаление невидимых частей объектов:

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

1. 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.

1. 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.

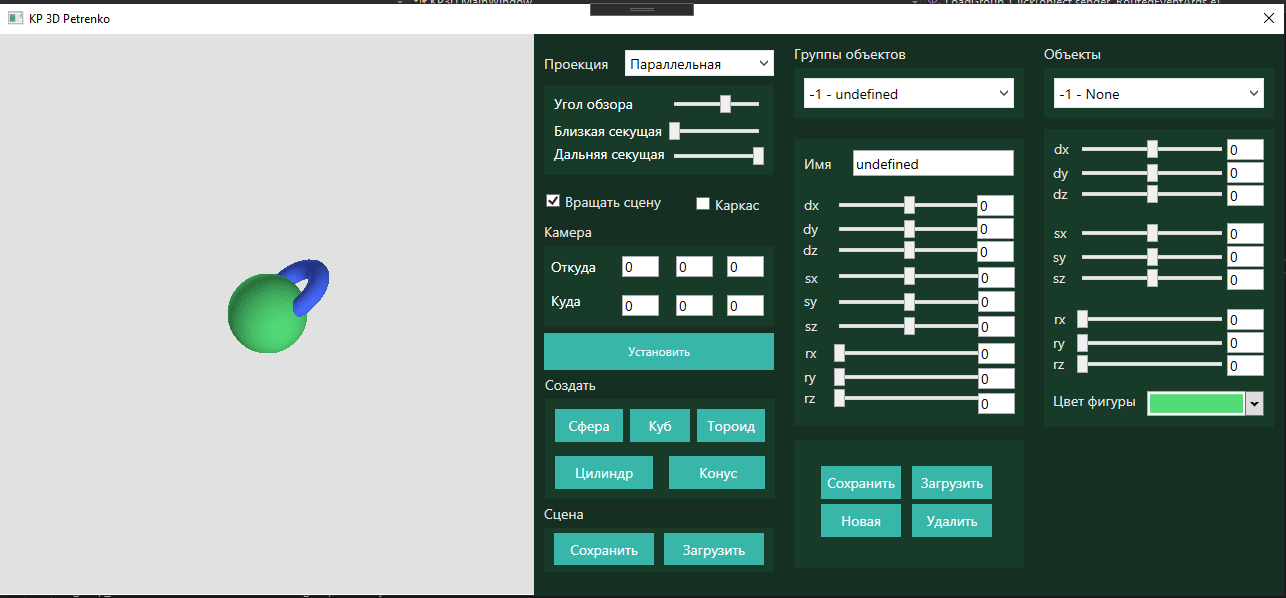


Рисунок Б.1 – Пример вращения объекта

Пример управления камерой приведен на рисунке Б.2.

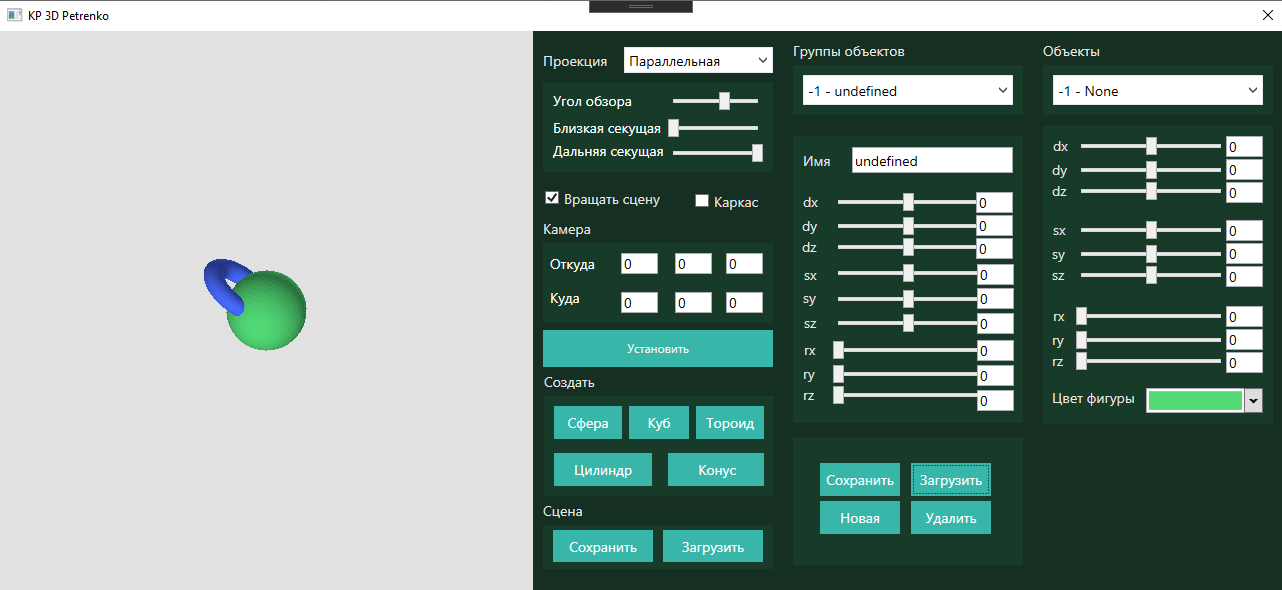


Рисунок Б.2 – Пример смещения камеры

# 

# Приложение Г. Листинг программы

using Ara3D;

using System;

using System.Collections.Generic;

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.CreateTranslation(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.CreatePerspectiveFieldOfView(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.Generic;

using System.Diagnostics;

using System.Drawing;

using System.Drawing.Imaging;

using System.Runtime.InteropServices;

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(ptrFirstPixel, 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.CreateRotationX(groups[g].shapes[i].rotation\_x);

var RotY = MyMath.MyMatrix4x4.CreateRotationY(groups[g].shapes[i].rotation\_y);

var RotZ = MyMath.MyMatrix4x4.CreateRotationZ(groups[g].shapes[i].rotation\_z);

var RotXG = MyMath.MyMatrix4x4.CreateRotationX(rotGX);

var RotYG = MyMath.MyMatrix4x4.CreateRotationY(rotGY);

var RotZG = MyMath.MyMatrix4x4.CreateRotationZ(rotGZ);

var Rot = RotX \* RotY \* RotZ;

var RotG = RotXG \* RotYG \* RotZG;

var Scale = MyMath.MyMatrix4x4.CreateScale(

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.CreateTranslation(

(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.CreateTranslation(

-(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.CreateFPSLookAt(camera.eye, camera.pitch, camera.yaw);

if (camera.center != Vector3.MinValue)

V = MyMath.MyMatrix4x4.CreateLookAt(camera.eye, camera.center, new Vector3(0, 1, 0));

float far = 100;

float near = 0.01f;

float A = 1 / (float)Math.Tan(camera.fov \* 0.5f);

float B = 1 / (float)Math.Tan(camera.fov \* 0.5f);

float C = far / (far - near);

float D = -1;

float E = near \* far / (far - near);

//var P = MyMath.MyMatrix4x4.CreateProjectionFOV(camera.fov, (float)width / (float)height, near, far);

var P = MyMath.MyMatrix4x4.CreateProjectionFOV(A, B, C, D, E);

Object \_lock = new Object();

Parallel.For(0, groups[g].shapes[i].triangles.Count, new ParallelOptions { MaxDegreeOfParallelism = 16 }, j =>

//for (int j = 0; j < shapes[i].triangles.Count; j++)

{

var p0 = MyMath.MyMatrix4x1.FromPoint4x1(groups[g].shapes[i].triangles[j].a);

var p1 = MyMath.MyMatrix4x1.FromPoint4x1(groups[g].shapes[i].triangles[j].b);

var p2 = MyMath.MyMatrix4x1.FromPoint4x1(groups[g].shapes[i].triangles[j].c);

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

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

var t3 = MyMath.MyMatrix4x1.ToPoint(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.Normalize().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].select\_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.AndYetAnotherMemesSavedTheWorld(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.Generic;

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.FromArgb(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 AndYetAnotherMemesSavedTheWorld(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+bd.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.Generic;

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.Generic;

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[bottomCapBaseIndex + 0] = rightIndex;

Tris[bottomCapBaseIndex + 1] = middleIndex;

Tris[bottomCapBaseIndex + 2] = leftIndex;

//assign top tris

Tris[topCapBaseIndex + 0] = topCapVertexOffset + leftIndex;

Tris[topCapBaseIndex + 1] = topCapVertexOffset + middleIndex;

Tris[topCapBaseIndex + 2] = topCapVertexOffset + rightIndex;

}

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

{

triangles.Add(new Shapes.MyTriangle(Vertices[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].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 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.Normalize() \* 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(vertices[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(triangles[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.Generic;

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(nbrRingSegments, 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.Generic;

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.Generic;

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.Generic;

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.Generic;

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.Imaging;

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.Serialization.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.selected\_gid].shapes[scene.selected\_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.Shape 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.Color.FromRgb(shape.main\_clr.R, shape.main\_clr.G, shape.main\_clr.B);

}

public void RefreshGroupsCB()

{

group\_selecter.Items.Clear();

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.SelectedIndex = 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.selected\_gid].shapes.Count; i++)

{

selecter.Items.Add($"{i} - " + scene.groups[scene.selected\_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.selected\_gid].shapes.Add(shape);

RefreshComboBox();

selecter.SelectedIndex = scene.groups[scene.selected\_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(object sender, MouseButtonEventArgs e)

{

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

int x = (int)e.GetPosition(SceneView).X;

int y = (int)e.GetPosition(SceneView).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.SelectedIndex = scene.selected\_gid + 1;

if (scene.selected\_id != -1)

GenerateShapeView(scene.groups[scene.selected\_gid].shapes[scene.selected\_id]);

if (scene.selected\_gid != -1)

LoadGroupInfo(scene.groups[scene.selected\_gid]);

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

}

BitmapImage BitmapToImageSource(Bitmap bitmap)

{

using (MemoryStream memory = new MemoryStream())

{

bitmap.Save(memory, System.Drawing.Imaging.ImageFormat.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(object 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(object 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.camera.yaw);

//Debug.WriteLine(scene.camera.pitch);

//Debug.WriteLine(scene.camera.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.selected\_gid].shapes.RemoveAt(scene.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.camera.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.ToString();

scene.selected\_id = int.Parse(a.Split(" ")[0]);

if (scene.selected\_id != -1)

GenerateShapeView(scene.groups[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(object sender, RoutedPropertyChangedEventArgs<System.Windows.Media.Color?> e)

{

if (scene.selected\_id != -1)

scene.groups[scene.selected\_gid].shapes[scene.selected\_id].main\_clr = Render.color(cp.SelectedColor.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(object sender, RoutedPropertyChangedEventArgs<double> e)

{

tbdx.Text = dx\_slider.Value.ToString();

if (scene.selected\_id != -1)

{

var shape = scene.groups[scene.selected\_gid].shapes[scene.selected\_id];

shape.dx = (float)dx\_slider.Value;

}

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

}

private void dy\_slider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

tbdy.Text = dy\_slider.Value.ToString();

if (scene.selected\_id != -1)

{

var shape = scene.groups[scene.selected\_gid].shapes[scene.selected\_id];

shape.dy = (float)dy\_slider.Value;

}

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

}

private void dz\_slider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

tbdz.Text = dz\_slider.Value.ToString();

if (scene.selected\_id != -1)

{

var shape = scene.groups[scene.selected\_gid].shapes[scene.selected\_id];

shape.dz = (float)dz\_slider.Value;

}

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

}

private void scx\_slider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

tbsx.Text = scx\_slider.Value.ToString();

if (scene.selected\_id != -1)

{

var shape = scene.groups[scene.selected\_gid].shapes[scene.selected\_id];

shape.scale\_x = (float)scx\_slider.Value;

}

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

}

private void scy\_slider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

tbsy.Text = scy\_slider.Value.ToString();

if (scene.selected\_id != -1)

{

var shape = scene.groups[scene.selected\_gid].shapes[scene.selected\_id];

shape.scale\_y = (float)scy\_slider.Value;

}

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

}

private void scz\_slider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

tbsz.Text = scz\_slider.Value.ToString();

if (scene.selected\_id != -1)

{

var shape = scene.groups[scene.selected\_gid].shapes[scene.selected\_id];

shape.scale\_z = (float)scz\_slider.Value;

}

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

}

private void rx\_slider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

tbrx.Text = rx\_slider.Value.ToString();

if (scene.selected\_id != -1)

{

var shape = scene.groups[scene.selected\_gid].shapes[scene.selected\_id];

shape.rotation\_x = (float)rx\_slider.Value;

}

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

}

private void ry\_slider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

tbry.Text = ry\_slider.Value.ToString();

if (scene.selected\_id != -1)

{

var shape = scene.groups[scene.selected\_gid].shapes[scene.selected\_id];

shape.rotation\_y = (float)ry\_slider.Value;

}

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

}

private void rz\_slider\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

tbrz.Text = rz\_slider.Value.ToString();

if (scene.selected\_id != -1)

{

var shape = scene.groups[scene.selected\_gid].shapes[scene.selected\_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(object sender, RoutedPropertyChangedEventArgs<double> e)

{

tbdx2.Text = dx\_slider2.Value.ToString();

if (scene.selected\_gid != -1)

{

var shape = scene.groups[scene.selected\_gid];

shape.dx = (float)dx\_slider2.Value;

}

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

}

private void dy\_slider\_ValueChanged2(object sender, RoutedPropertyChangedEventArgs<double> e)

{

tbdy2.Text = dy\_slider2.Value.ToString();

if (scene.selected\_gid != -1)

{

var shape = scene.groups[scene.selected\_gid];

shape.dy = (float)dy\_slider2.Value;

}

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

}

private void dz\_slider\_ValueChanged2(object sender, RoutedPropertyChangedEventArgs<double> e)

{

tbdz2.Text = dz\_slider2.Value.ToString();

if (scene.selected\_gid != -1)

{

var shape = scene.groups[scene.selected\_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, RoutedPropertyChangedEventArgs<double> e)

{

tbsx2.Text = scx\_slider2.Value.ToString();

if (scene.selected\_gid != -1)

{

var shape = scene.groups[scene.selected\_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, RoutedPropertyChangedEventArgs<double> e)

{

tbsy2.Text = scy\_slider2.Value.ToString();

if (scene.selected\_gid != -1)

{

var shape = scene.groups[scene.selected\_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, RoutedPropertyChangedEventArgs<double> e)

{

tbsz2.Text = scz\_slider2.Value.ToString();

if (scene.selected\_gid != -1)

{

var shape = scene.groups[scene.selected\_gid];

shape.scale\_z = (float)scz\_slider2.Value;

}

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

}

private void rx\_slider\_ValueChanged2(object sender, RoutedPropertyChangedEventArgs<double> e)

{

tbrx2.Text = rx\_slider2.Value.ToString();

if (scene.selected\_gid != -1)

{

var shape = scene.groups[scene.selected\_gid];

shape.rotation\_x = (float)rx\_slider2.Value;

}

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

}

private void ry\_slider\_ValueChanged2(object sender, RoutedPropertyChangedEventArgs<double> e)

{

tbry2.Text = ry\_slider2.Value.ToString();

if (scene.selected\_gid != -1)

{

var shape = scene.groups[scene.selected\_gid];

shape.rotation\_y = (float)ry\_slider2.Value;

}

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

}

private void rz\_slider\_ValueChanged2(object sender, RoutedPropertyChangedEventArgs<double> e)

{

tbrz2.Text = rz\_slider2.Value.ToString();

if (scene.selected\_gid != -1)

{

var shape = scene.groups[scene.selected\_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.selected\_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.Count; 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(clr[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.SelectedIndex = 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\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (group\_selecter.Items.Count > 0)

{

var a = group\_selecter.SelectedItem.ToString();

scene.selected\_gid = int.Parse(a.Split(" ")[0]);

if (scene.selected\_gid != -1)

{

LoadGroupInfo(scene.groups[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(scene.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.selected\_gid].shapes.Count; i++)

{

fs.WriteLine(scene.groups[scene.selected\_gid].shapes[i].type);

fs.WriteLine(scene.groups[scene.selected\_gid].shapes[i].dx);

fs.WriteLine(scene.groups[scene.selected\_gid].shapes[i].dy);

fs.WriteLine(scene.groups[scene.selected\_gid].shapes[i].dz);

fs.WriteLine(scene.groups[scene.selected\_gid].shapes[i].scale\_x);

fs.WriteLine(scene.groups[scene.selected\_gid].shapes[i].scale\_y);

fs.WriteLine(scene.groups[scene.selected\_gid].shapes[i].scale\_z);

fs.WriteLine(scene.groups[scene.selected\_gid].shapes[i].rotation\_x);

fs.WriteLine(scene.groups[scene.selected\_gid].shapes[i].rotation\_y);

fs.WriteLine(scene.groups[scene.selected\_gid].shapes[i].rotation\_z);

fs.WriteLine(

scene.groups[scene.selected\_gid].shapes[i].main\_clr.R

+ " "

+ scene.groups[scene.selected\_gid].shapes[i].main\_clr.G

+ " "

+ scene.groups[scene.selected\_gid].shapes[i].main\_clr.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.selected\_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(clr[0]), int.Parse(clr[1]), int.Parse(clr[2]));

scene.groups[scene.selected\_gid].shapes.Add(shape);

}

}

}

RefreshGroupsCB();

group\_selecter.SelectedIndex = scene.groups.Count;

}

}

}

}

# Приложение Д. СПРАВКА АНТИПЛАГИАТА

