МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Южно-Уральский государственный университет (национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

Разработка демонстрационного комплекса для алгоритмов загораживания Робертса, Аппеля, Z-буфера

Руководитель работы:
к.т.н., доцент каф. ЭВМ
Ярош Е.С.

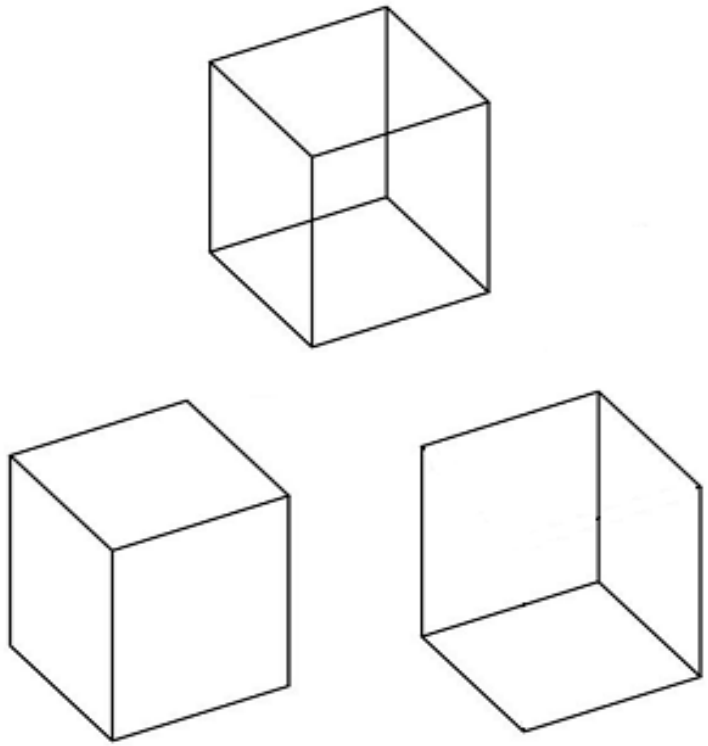
Автор работы:
студент КЭ-405
Росс К.А.

Актуальность работы

Современная компьютерная графика – сложная и разнообразная научно-техническая дисциплина.

Большинство ее алгоритмов содержат под собой достаточно подробную теоретическую основу.

Имеет смысл визуализировать работу отдельных алгоритмов с помощью технологий компьютерного моделирования.



Необходимость удаления невидимых линий

Цель и задачи проекта

Цель:

Разработать демонстрационный программный комплекс для изучения алгоритмов загораживания. В приложение необходимо включить алгоритмы Робертса, Аппеля, Z-буфера.

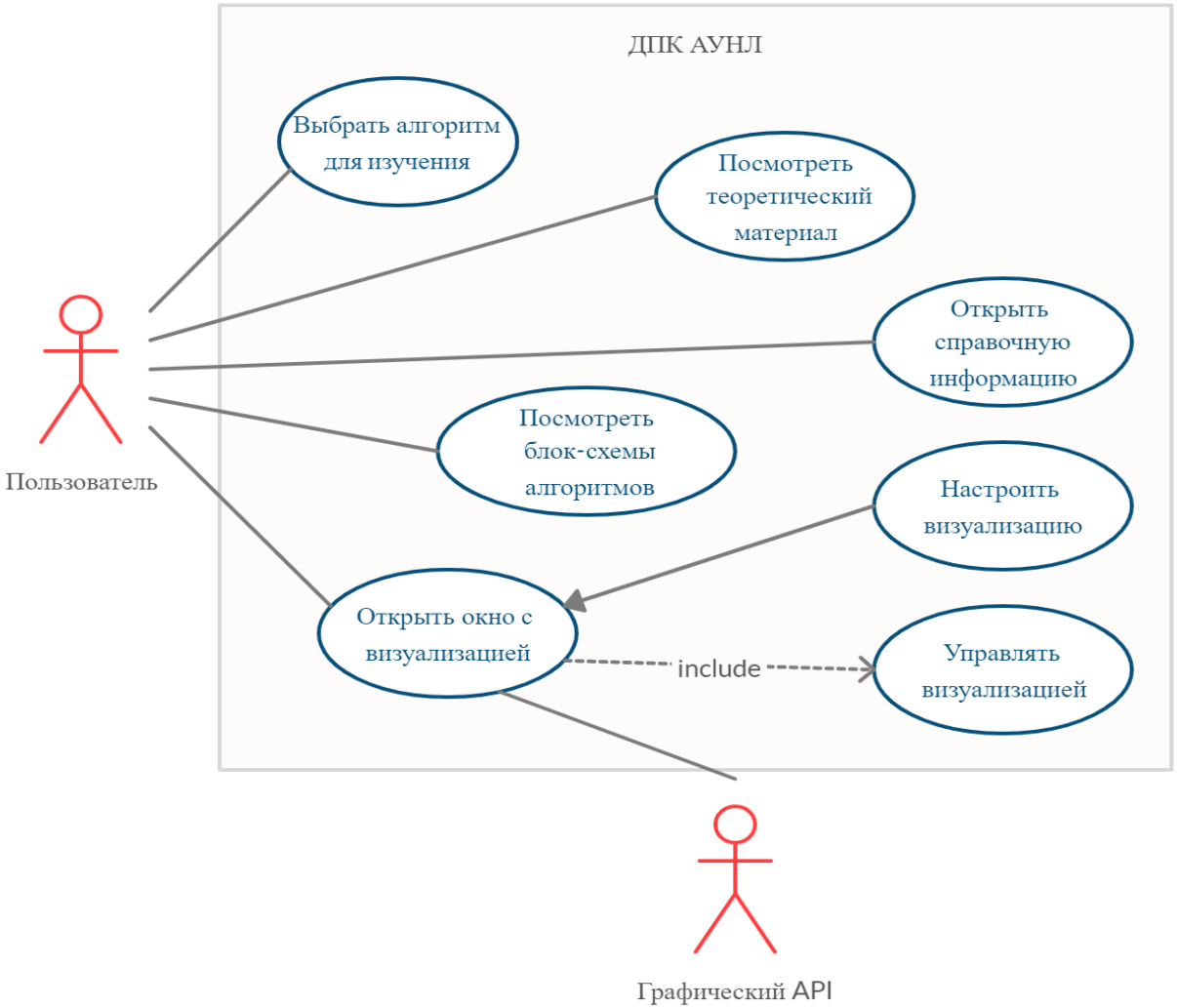
Задачи:

1. Рассмотреть алгоритмы загораживания, указанные в задании.
2. Сделать обзор существующих аналогов, демонстрирующих работу алгоритмов.
3. Выбрать инструменты для разработки системы.
4. Разработать архитектуру собственного программного комплекса.
5. Выполнить программную реализацию комплекса.
6. Оценить работоспособность готовой системы.

Аналоги

Наименование	Преимущества	Недостатки
1. Теоретические материалы	• информация по всем алгоритмам загораживания.	• отсутствие демонстрации применения.
2. Программный пакет на языке C++ (содержит Алгоритм Робертса)	• визуальное представление; • помимо алгоритмов загораживания, есть другие алгоритмы компьютерной графики.	• нет теоретического обоснования; • нет настроек визуализации; • алгоритм Робертса применяется только для одного объекта в сцене.
3. Алгоритм Z-буфера с применением технологии WebGL	• визуальное представление; • смена точки наблюдения; • открытый программный код.	• нет теоретического обоснования; • нет поэтапной визуализации алгоритма; • отсутствуют настройки визуализации.
4. Программный пакет на языке Java	• визуальное представление; • настройка процесса визуализации, включая задание параметров.	• нет теоретического обоснования; • нет поэтапной визуализации алгоритма.

Функционал системы

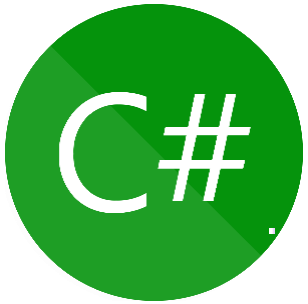


Отличительные особенности новой системы:

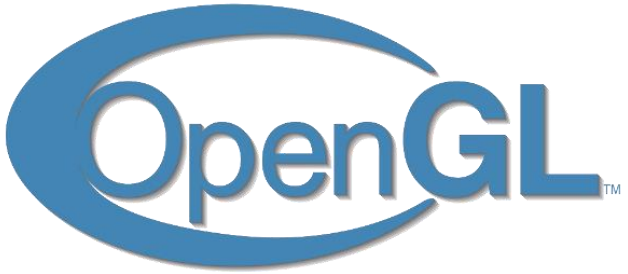
- алгоритмы расположены в одном месте;
- наличие раздела с теоретическим материалом, блок-схемами;
- возможность настройки визуализации;
- возможность манипулирования процессом визуализации.

Средства реализации

Среда разработки: Visual Studio 2017 Community Edition.

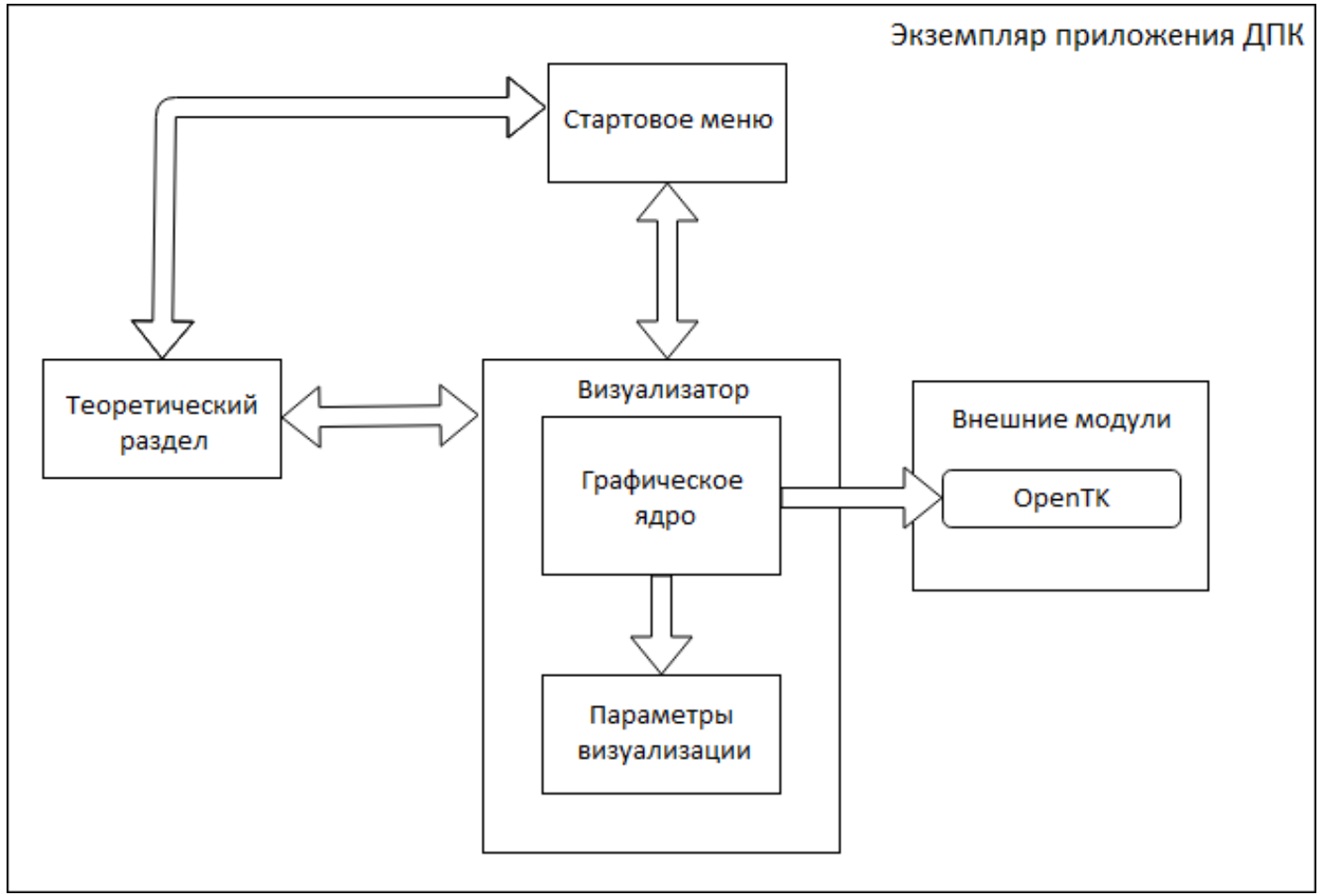


Язык программирования: C#.



Используемые библиотеки: OpenGL под враппером OpenTK (библиотека для работы с графикой), Windows Forms.

Архитектура системы



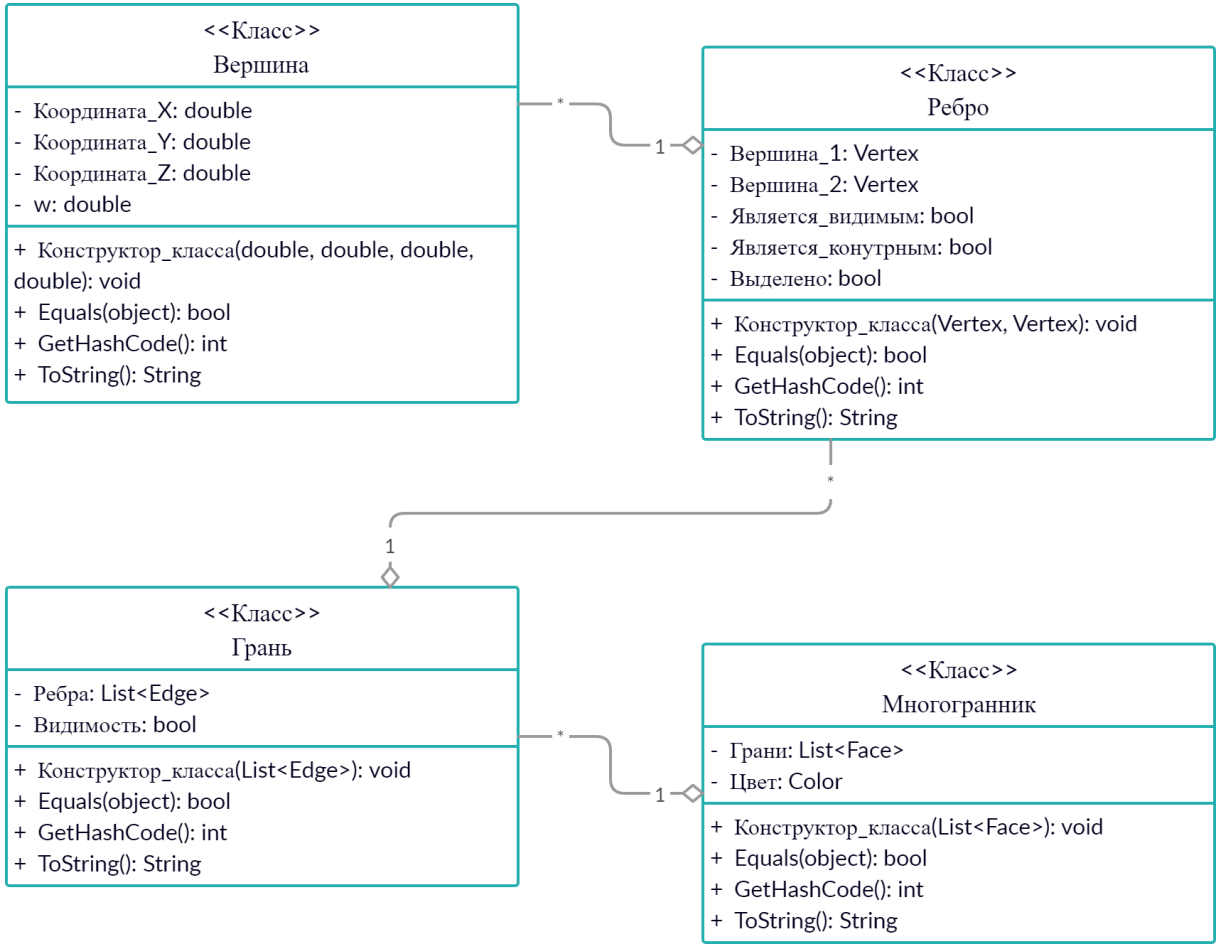
Теоретический раздел и визуализатор – основные модули приложения.

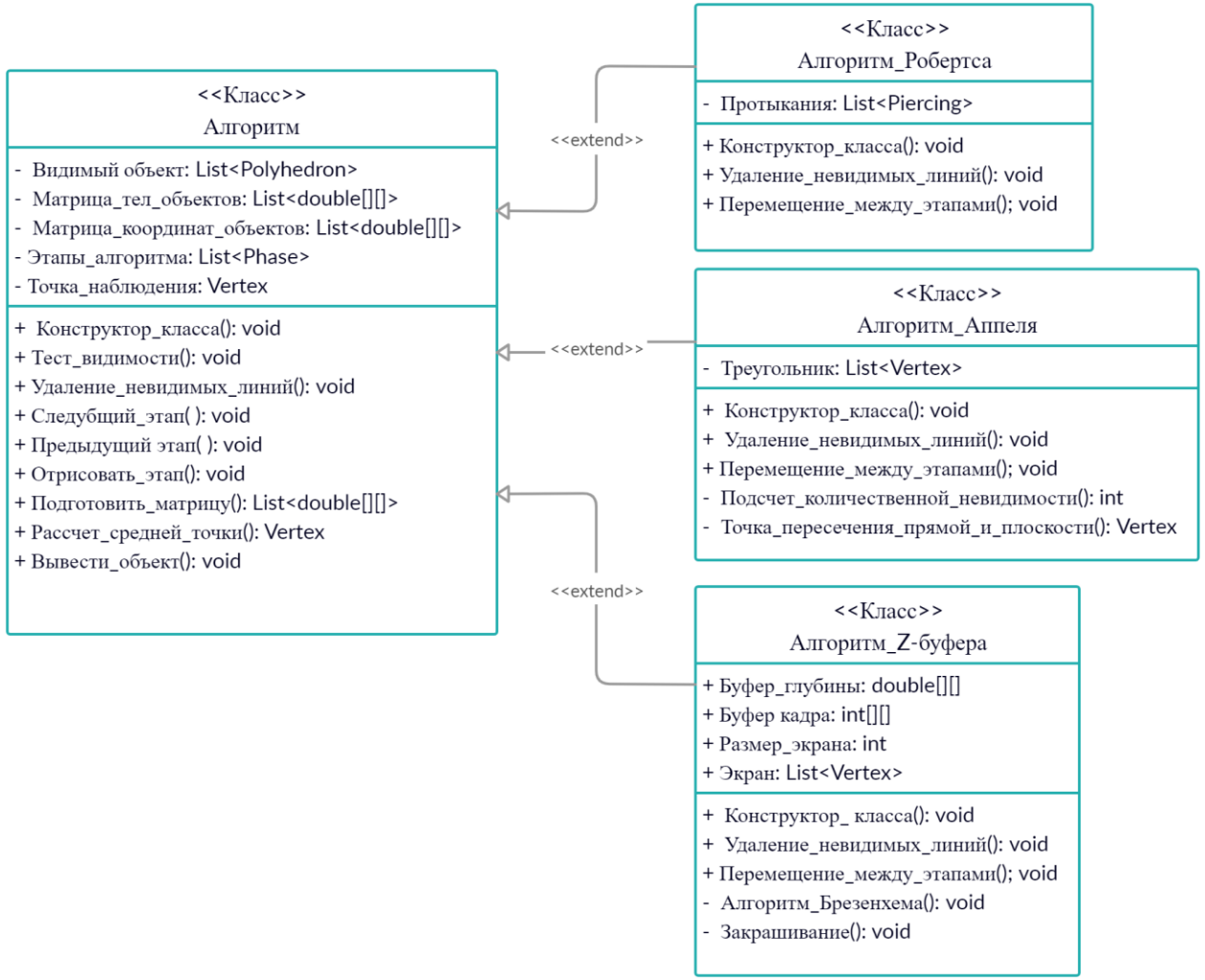
Дополнительные модули:

- стартовое меню;
- настройки визуализации.

Каждый из модулей выполнен в отдельном окне Windows.

Классы для алгоритмов загораживания

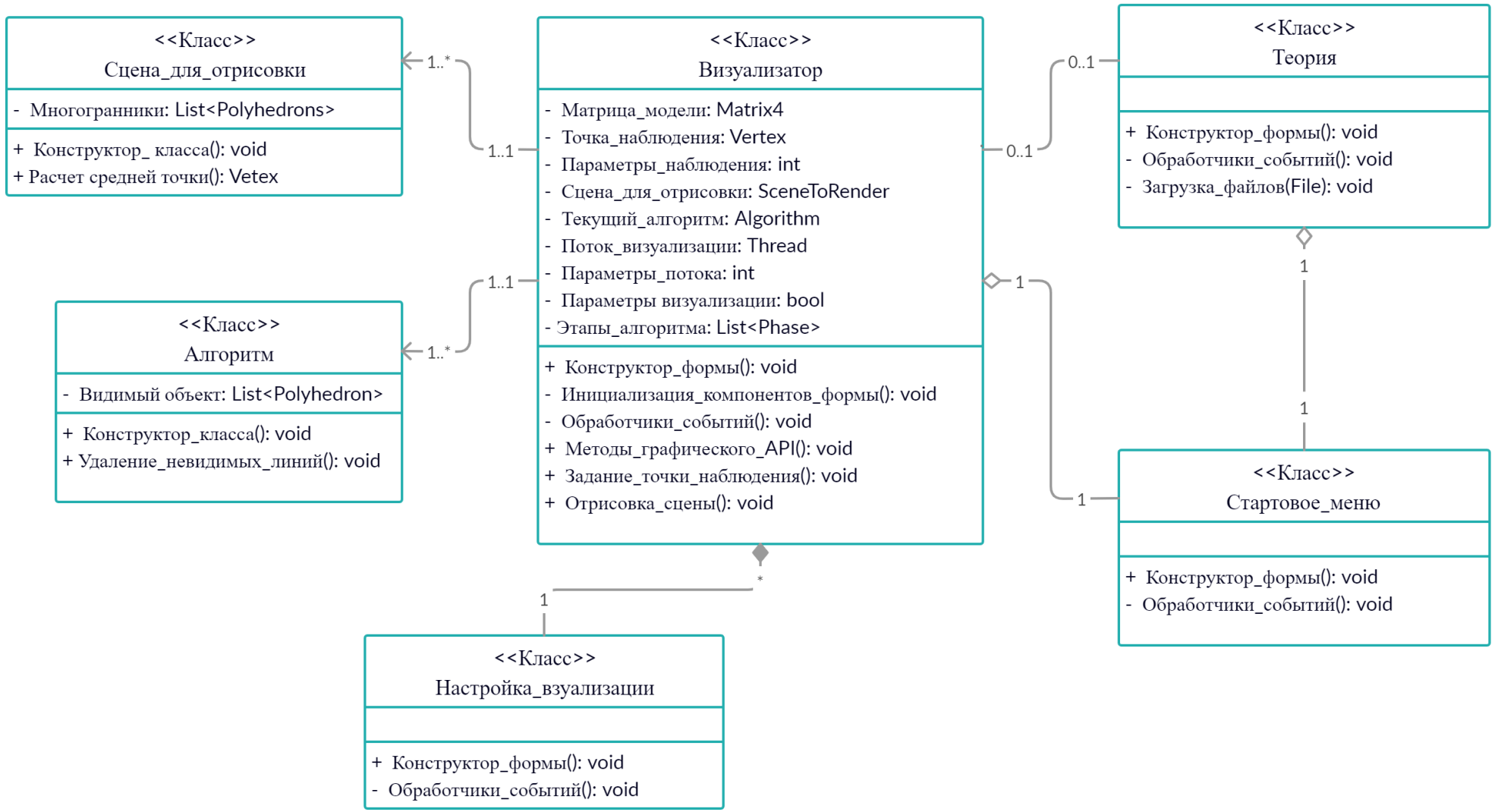




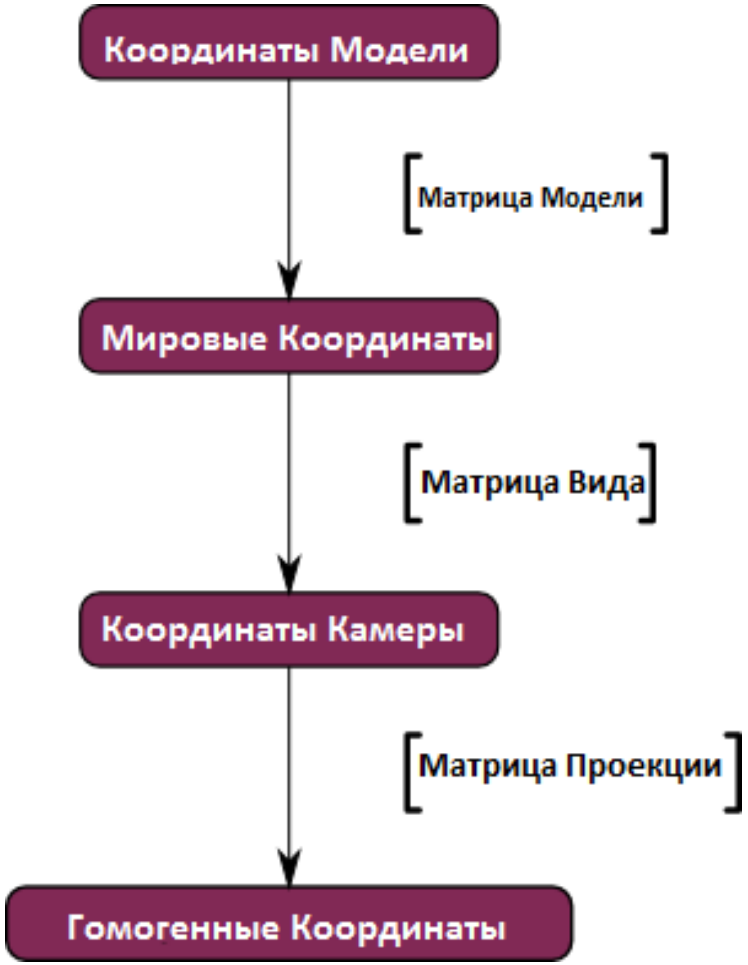
Графические примитивы

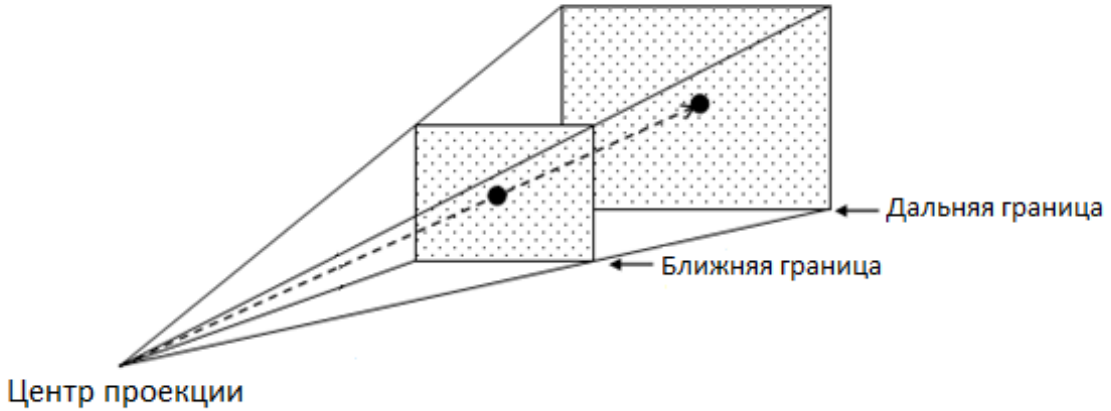
Алгоритмы загораживания

Общая диаграмма классов



Компьютерное моделирование объекта

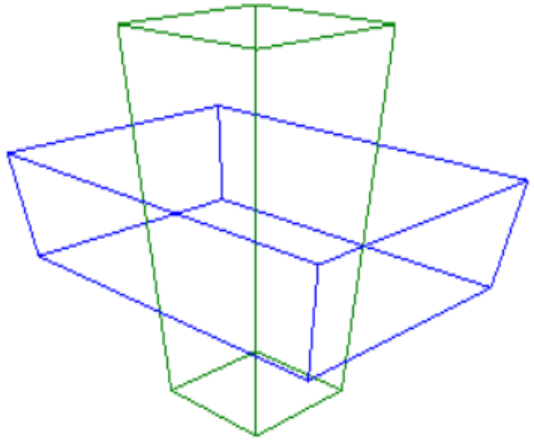




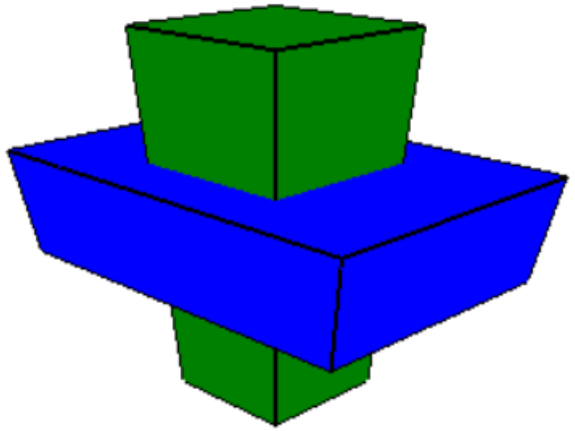
Пирамида охвата видимости

В процессе создания сцены используются несколько координатных систем, т.к. некоторые операции в них становятся проще для исполнения.

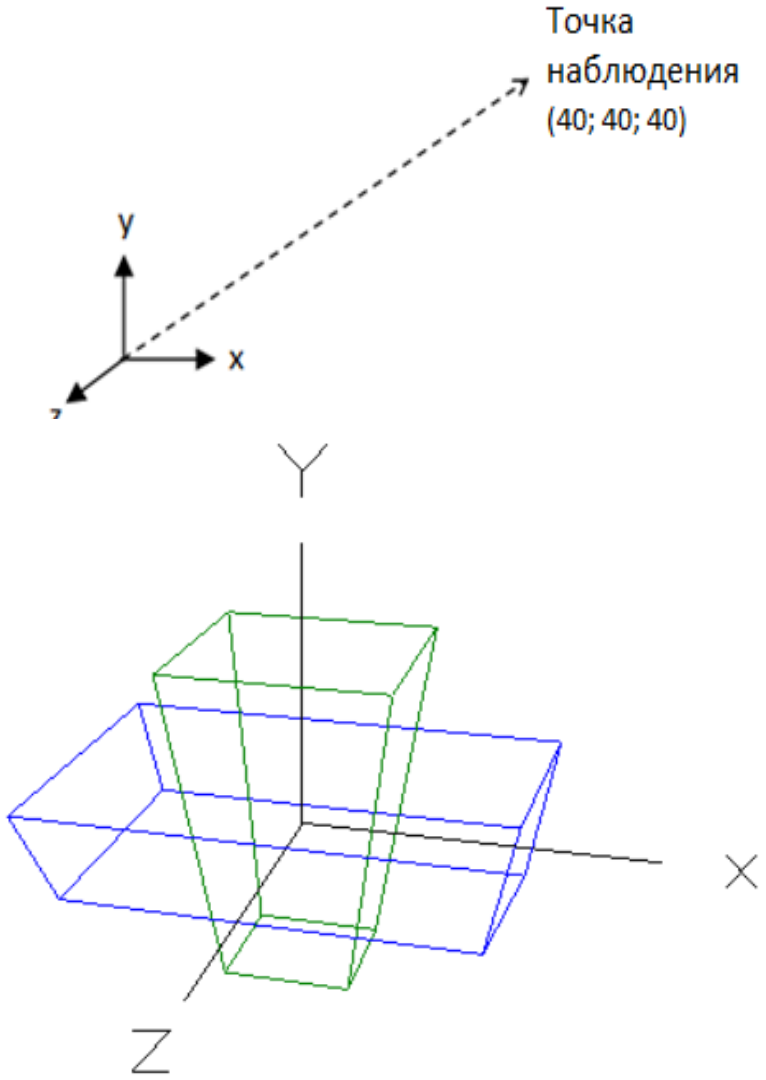
Тестовая сцена



Каркасное изображение



Желаемый результат



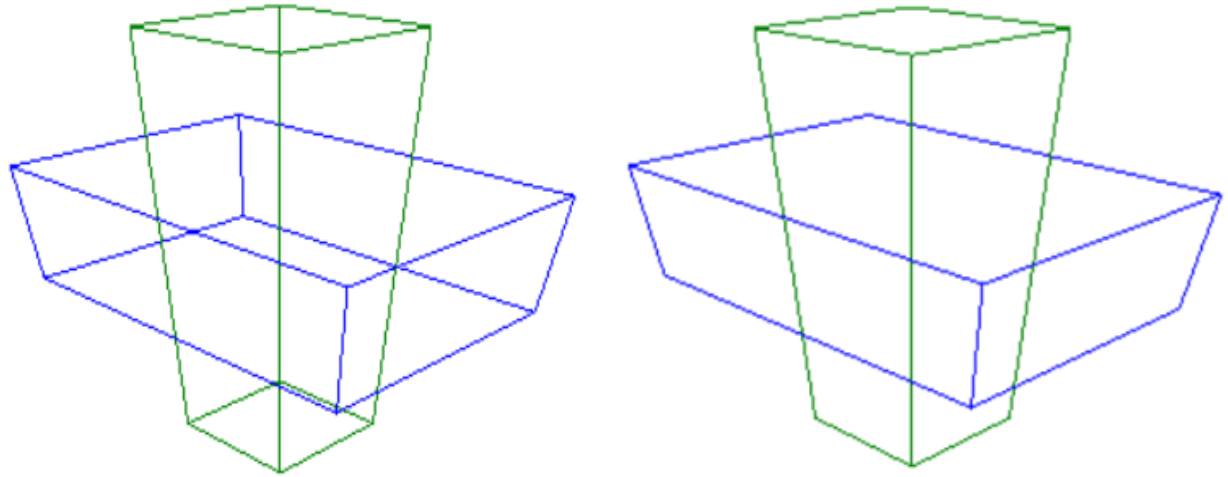
Система координат

Тест видимости

При выполнении теста видимости:

– определяется нормаль к грани;

– строится линия визирования, т.е. прямая, проходящая через точку наблюдения и основание нормали.



Применение теста видимости для тестовой сцены

Если угол между нормалью и линией визирования меньше $\pi/2$ – грань потенциально видима. Если угол больше $\pi/2$ – невидима.

Алгоритм Робертса

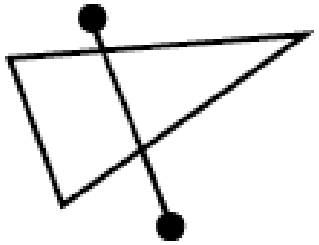
Работа Алгоритм Робертса проходит в две стадии:

1. Определение невидимых граней для каждого тела в сцене отдельно (тест видимости).
2. Определение и удаление невидимых ребер, которые экранируются гранями других объектов.

Возможны следующие случаи расположения:

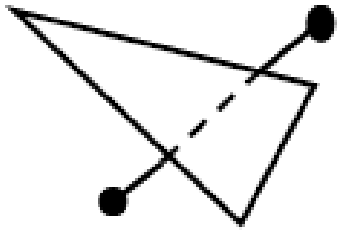
1. Грань ребра не закрывает. *Ребро остается в списке ребер.*
2. Грань полностью закрывает ребро. *Ребро удаляется из списка рассматриваемых ребер.*
3. Грань частично закрывает ребро. *В этом случае ребро разбивается на несколько частей.*

1)

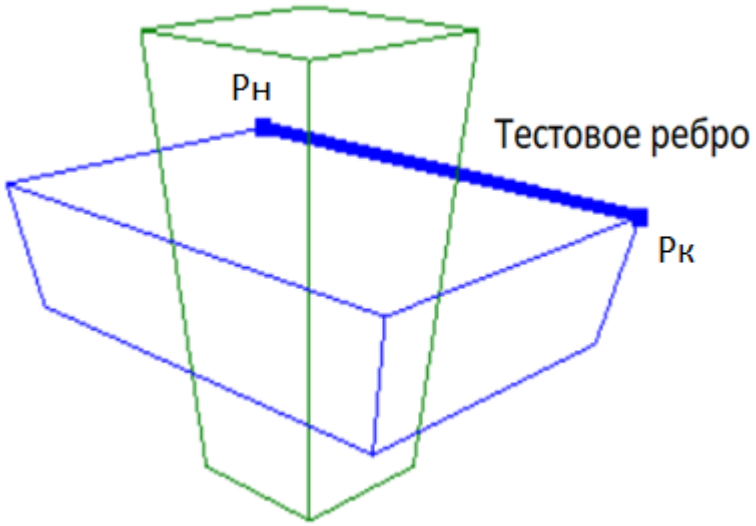


2)

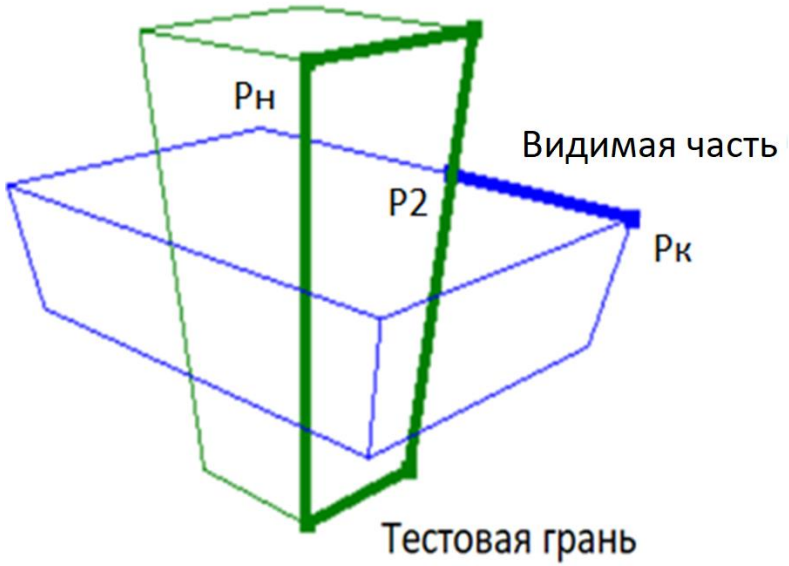
3)



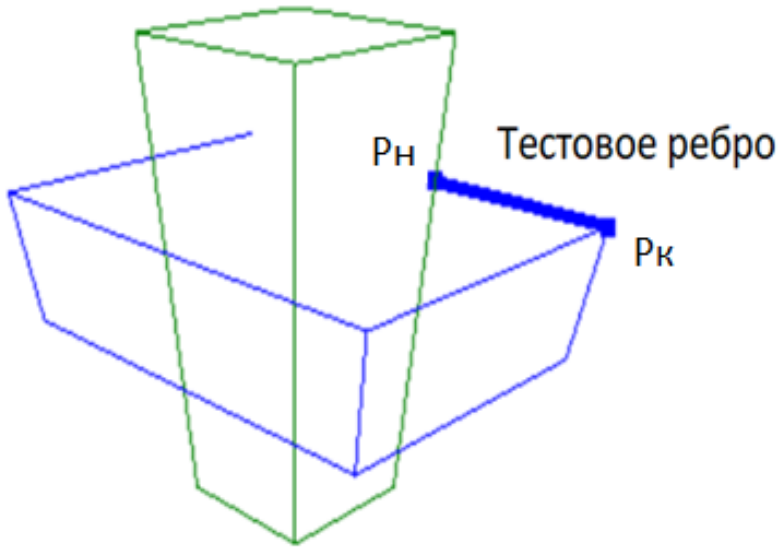
Алгоритм Робертса



Выбор тестового ребра

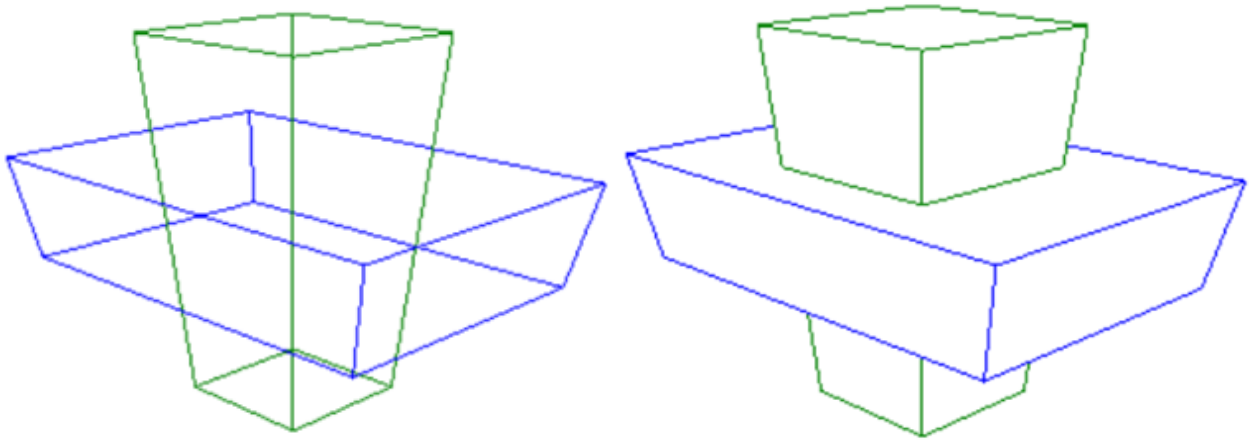


Определение видимой части



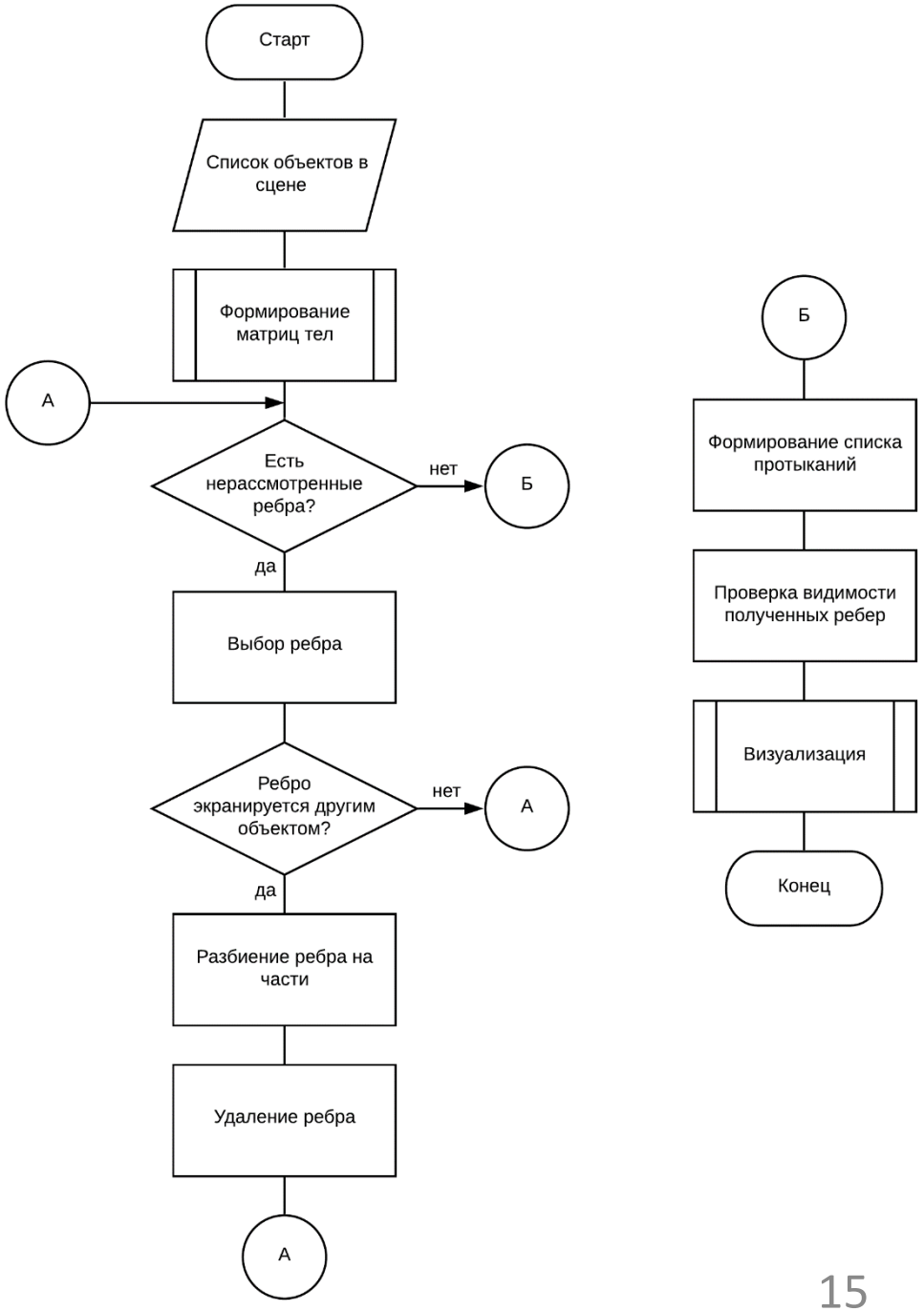
Результат

Алгоритм Робертса



Применение алгоритма Робертса
к тестовой сцене

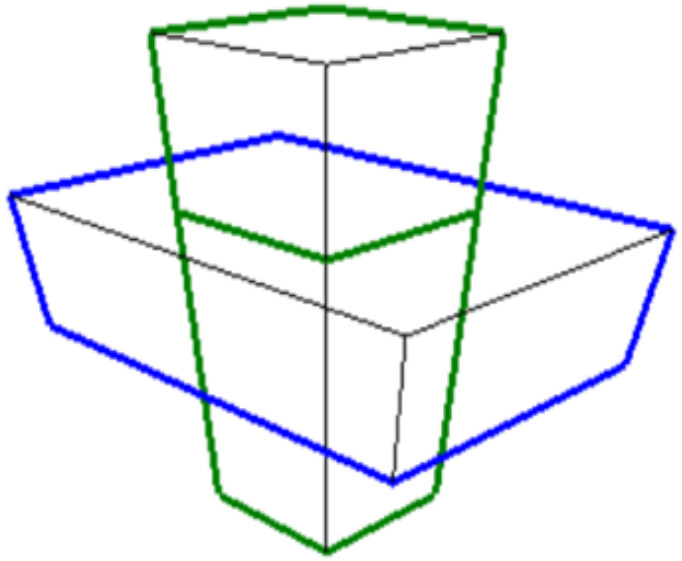
Временные затраты алгоритма пропорциональны квадрату числа ребер объектов в сцене ($O(n^2)$).



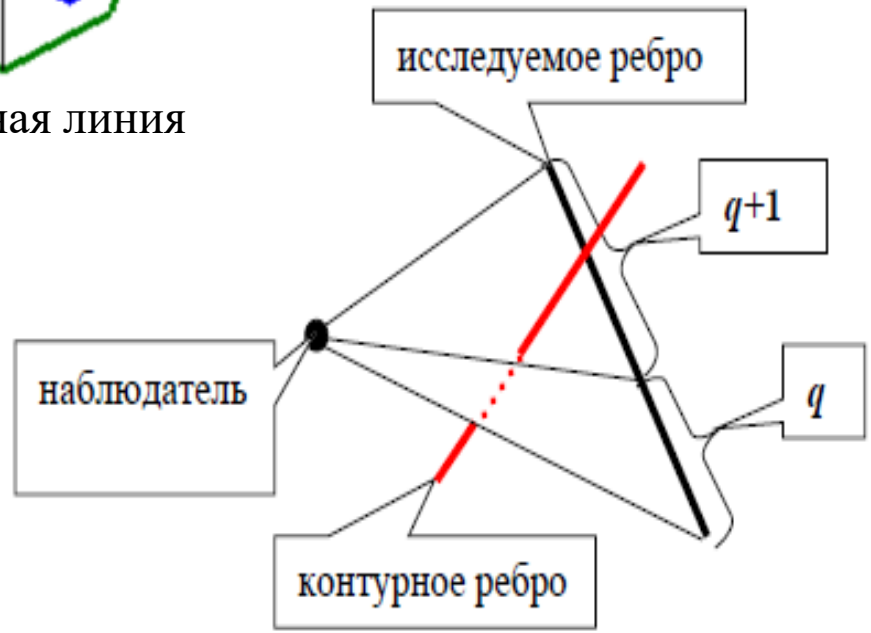
Алгоритм Аппеля

Количественная невидимость точки (Q) – число лицевых граней, закрывающих данную точку поверхности. Точка является видимой в том случае, если ее количественная невидимость равна нулю.

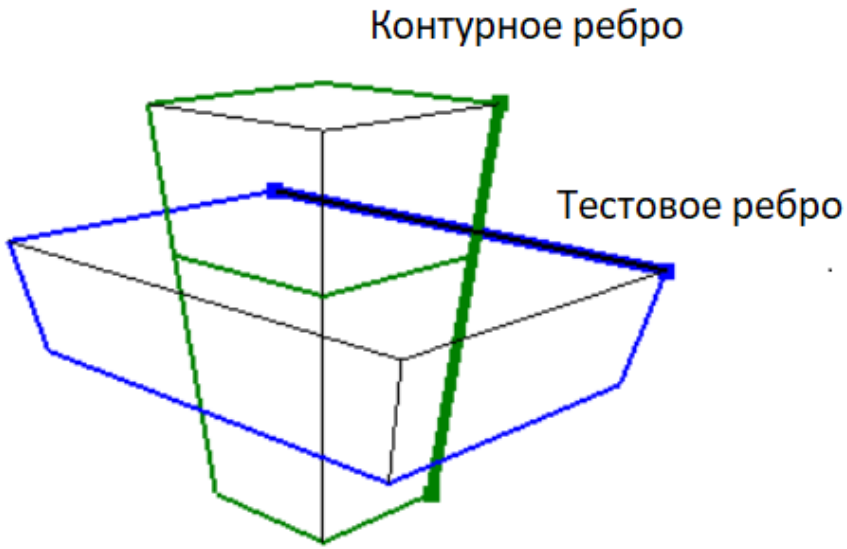
Количественная невидимость точки ребра увеличивается на 1 если ребро уходит за контурное ребро, и уменьшается на 1, если ребро выходит из-за контурного ребра.

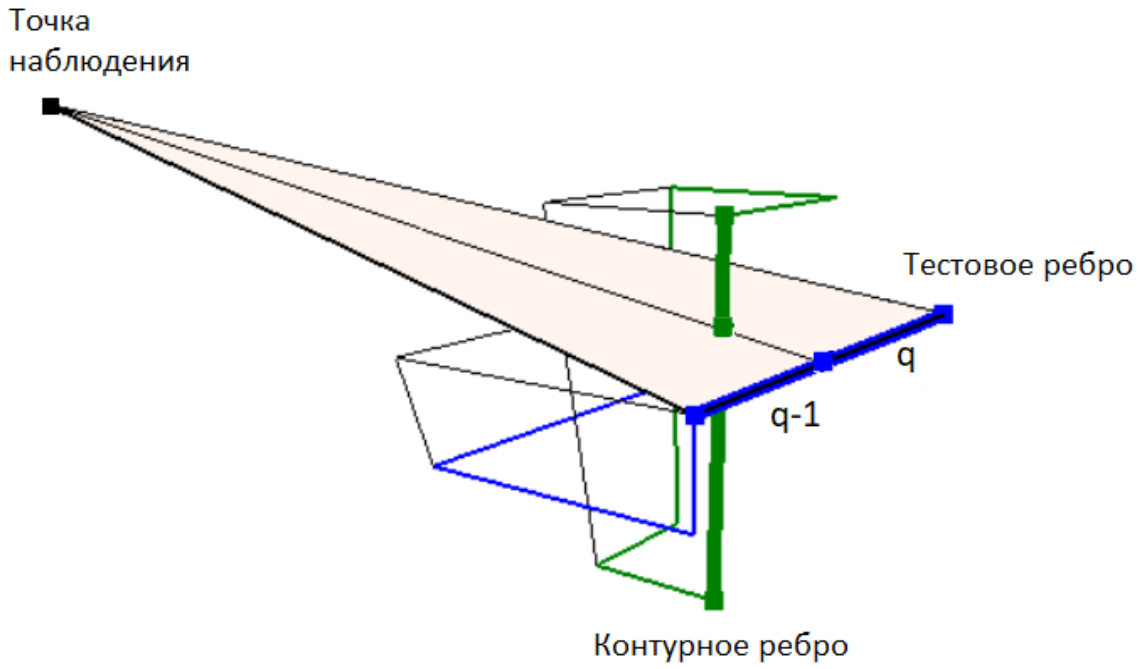


Контурная линия



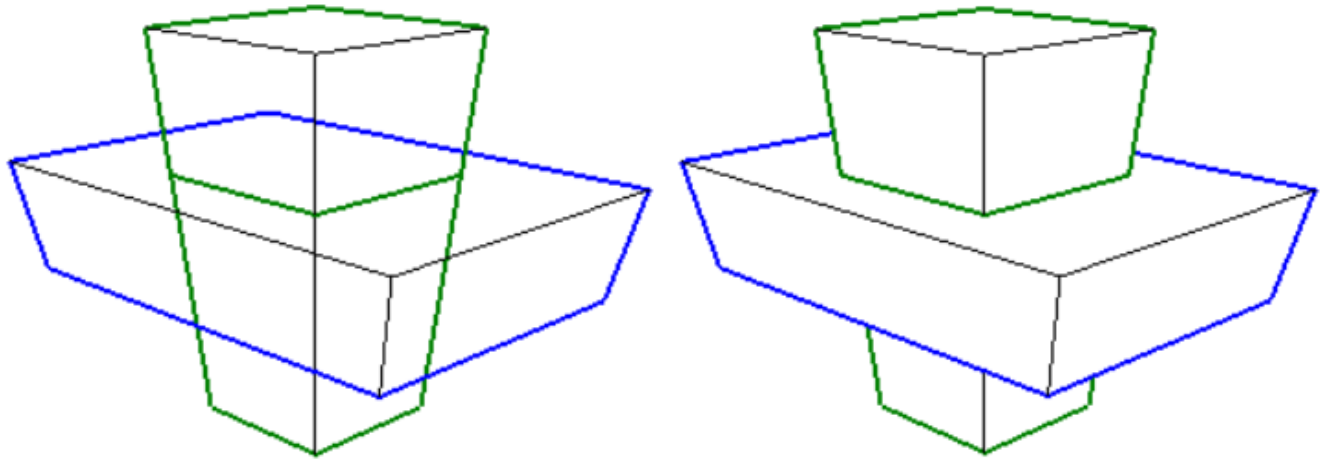
Алгоритм Аппеля



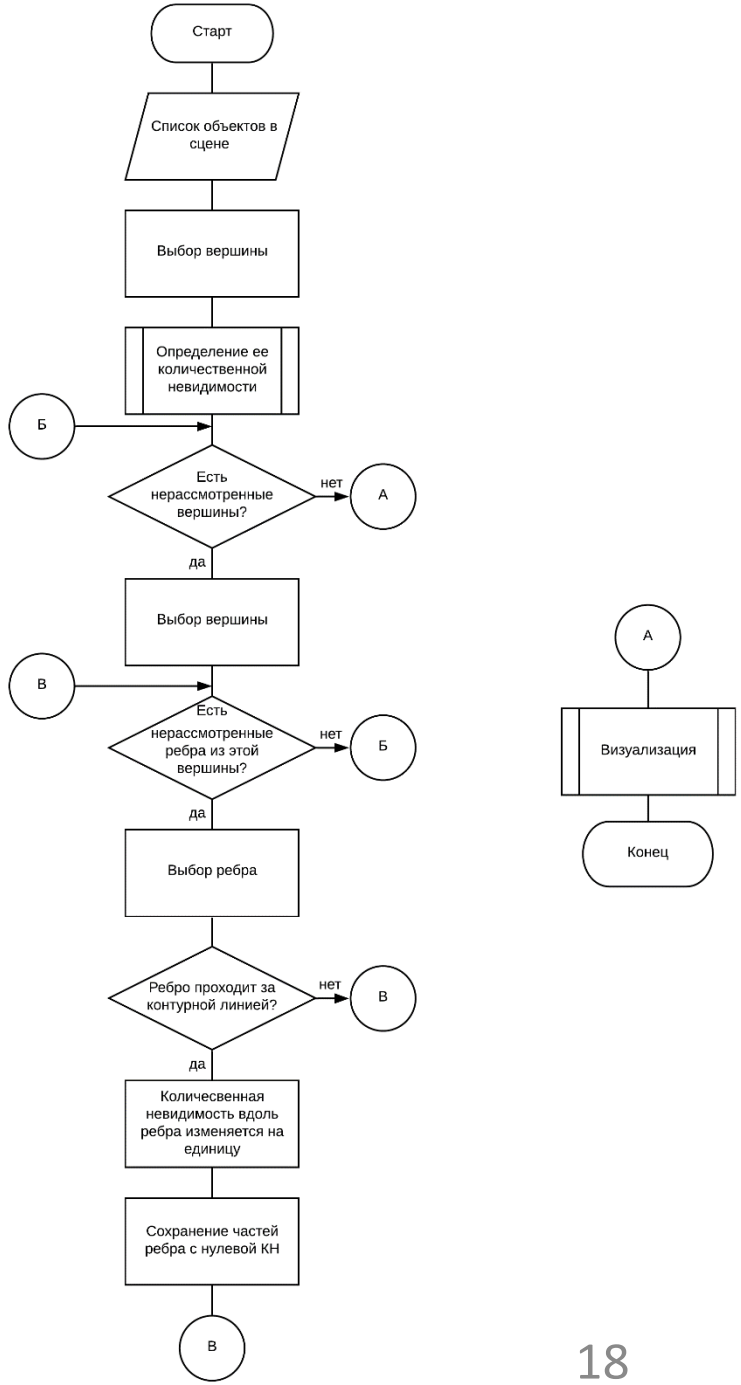


Количественная невидимость тестового ребра изменяется на 1, т.к. точка пересечения контурного ребра с плоскостью треугольника лежит внутри него, т.е. ребро протыкает треугольник.

Алгоритм Аппеля



Применение алгоритма Аппеля
к тестовой сцене



По сравнению с алгоритмом Робертса алгоритм Аппеля более быстр (сложность $O\left(\sqrt{n}\right)$, где n – общее число ребер), т.к. число ребер, входящих в контурную линию, меньше общего числа ребер в сцене.

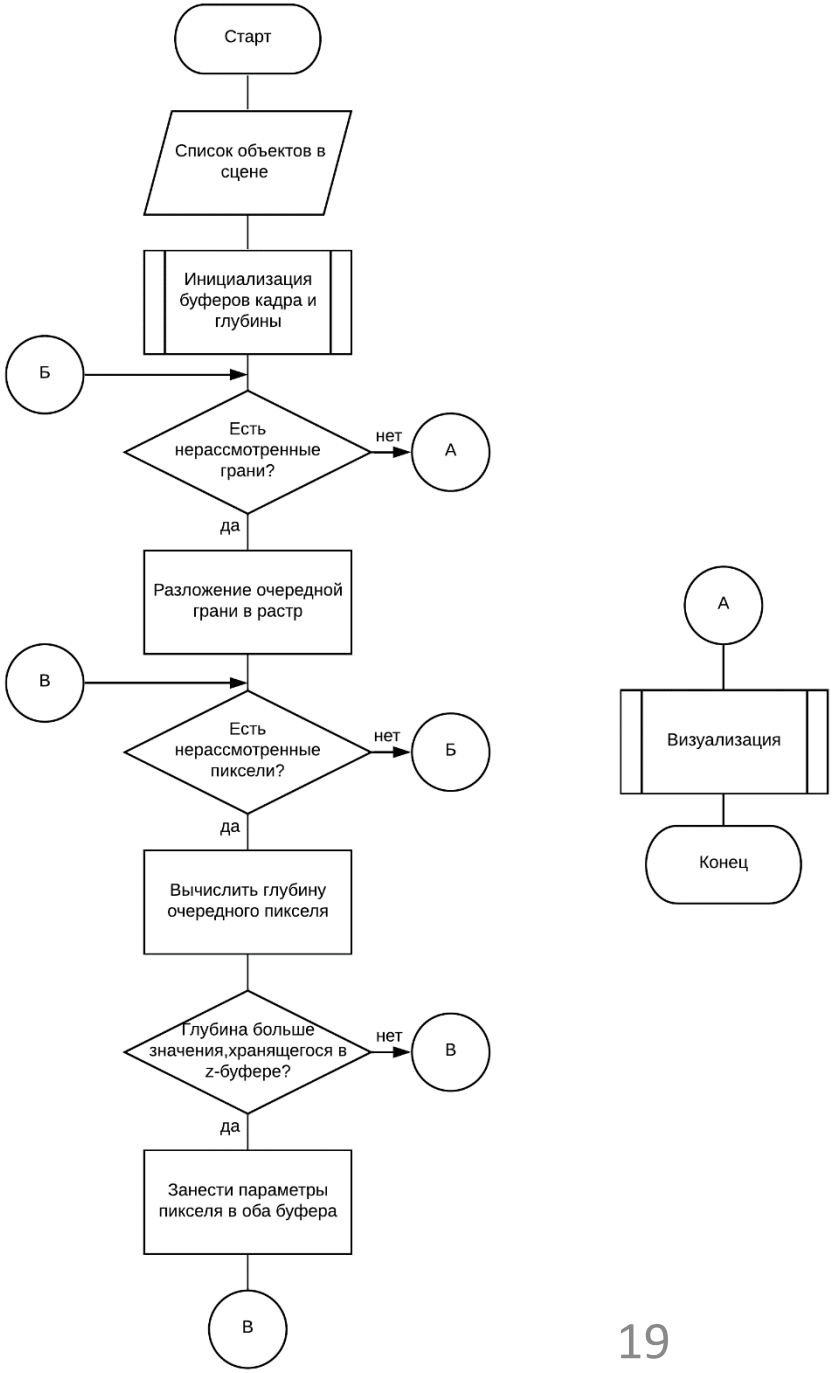
Алгоритм Z-буфера

Используются два буфера:

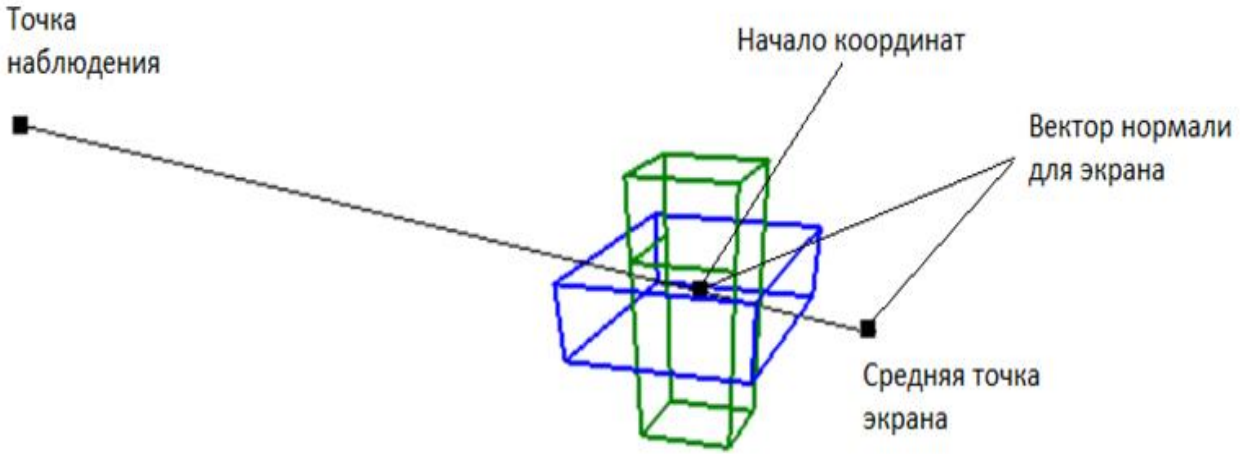
– буфер кадра – для запоминания цвета каждого пикселя в пространстве изображения;

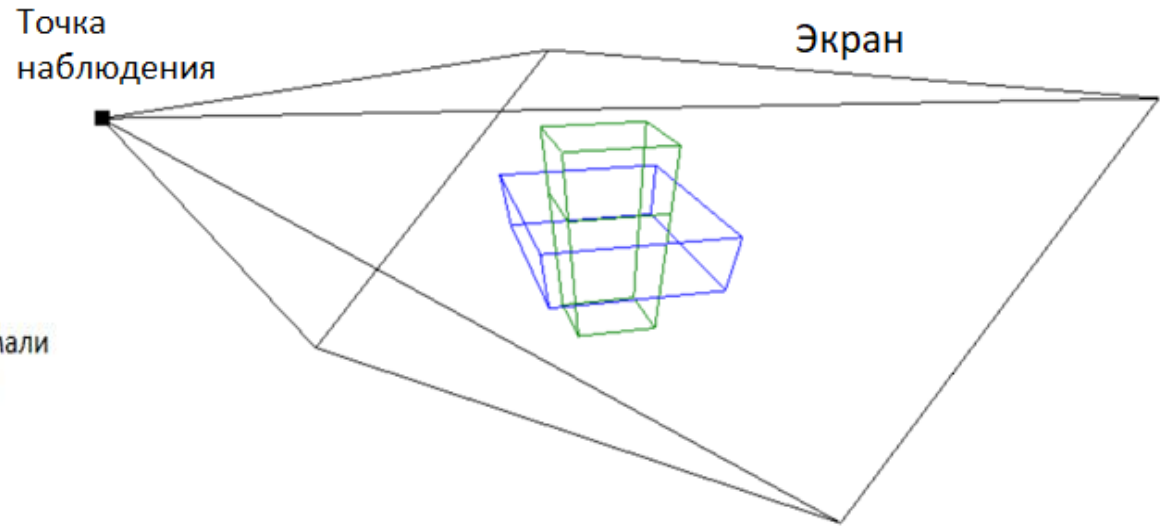
– *z*-буфер или буфер глубины – для запоминания координаты *z* или глубины каждого видимого пикселя в пространстве изображения.

Если *z*-координата нового пикселя больше значения буфера, то он ближе к наблюдателю и, следовательно, видим. Его характеристики заносятся в буфер кадра и *z*-буфер.

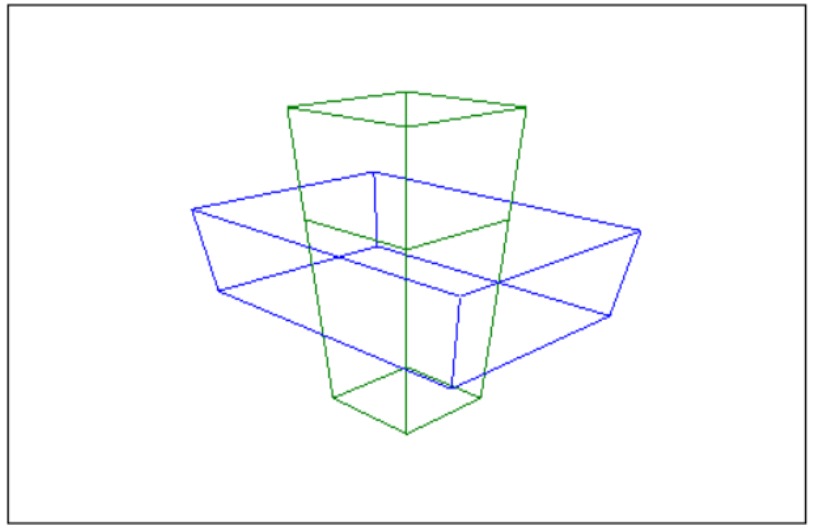


Алгоритм Z-буфера



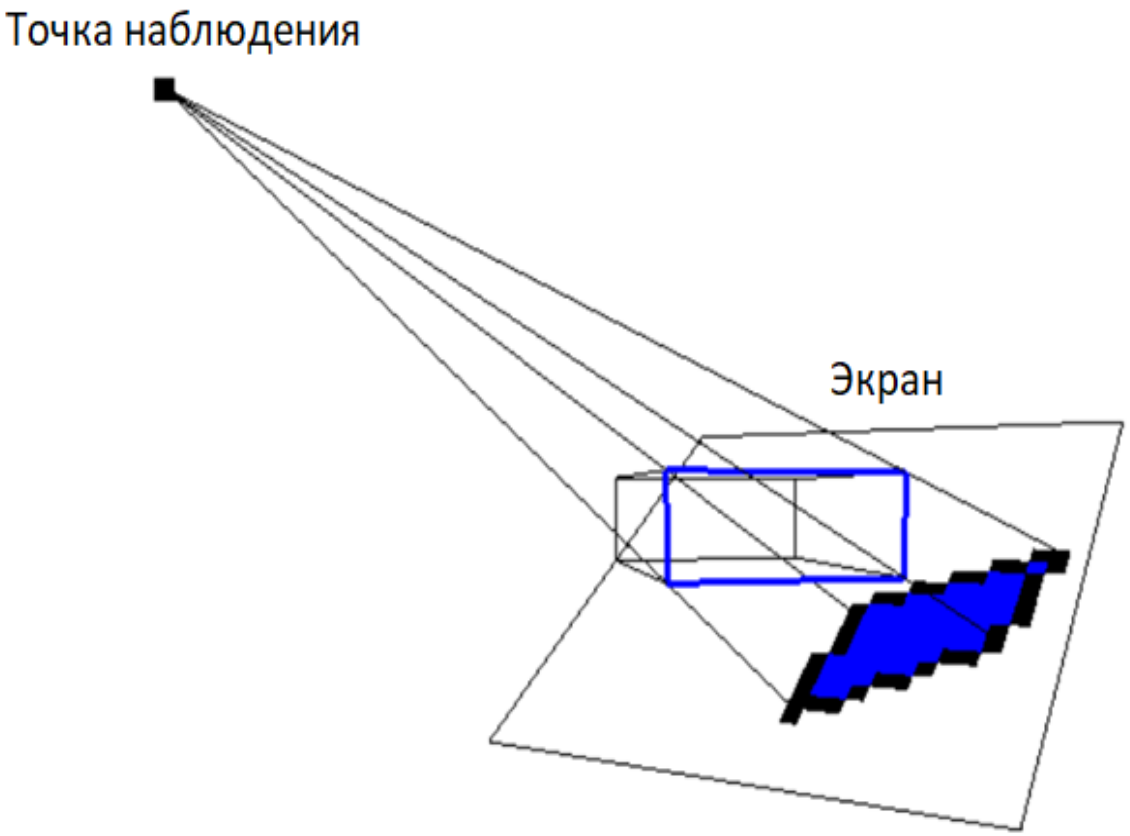


Область, на которую будут проецироваться сцены, перпендикулярна линии визирования и отступает от начала координат на заданный отрезок.

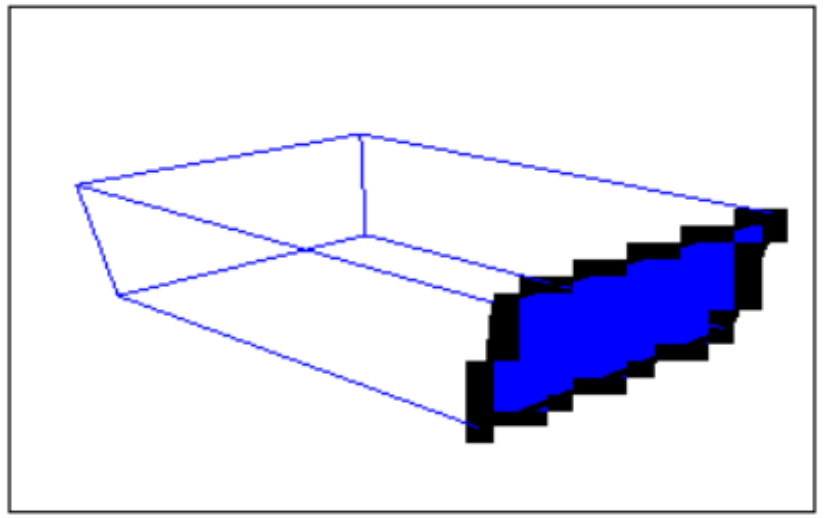


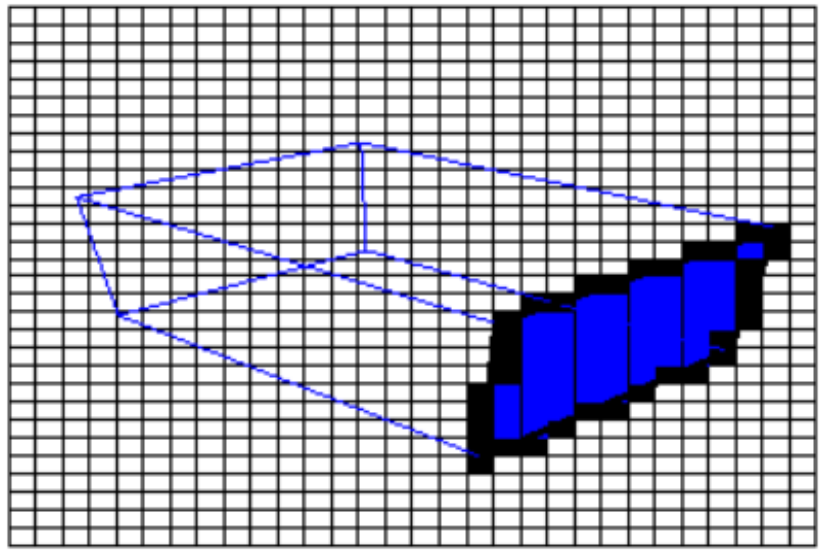
Положение точки наблюдения

Алгоритм Z-буфера



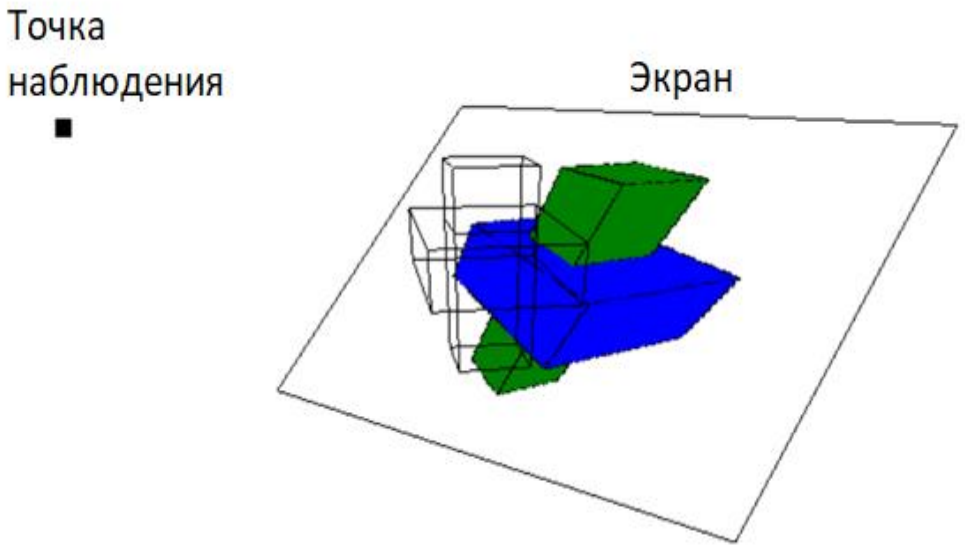
Центральное проецирование грани





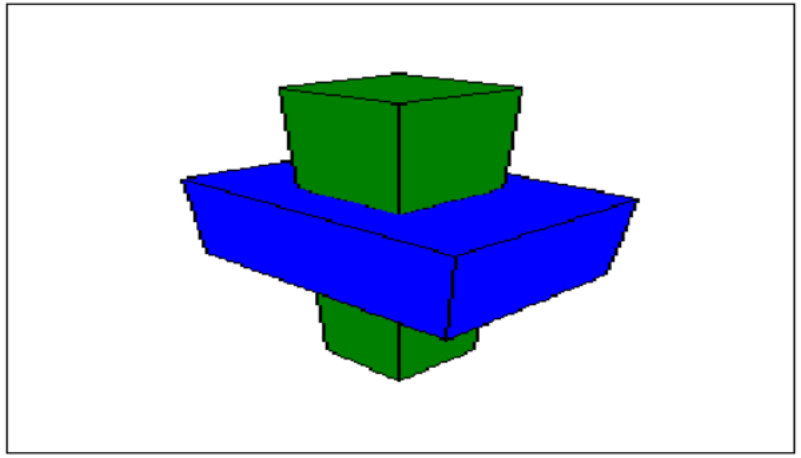
Разрешение экрана 30×30 пикселей

Алгоритм Z-буфера

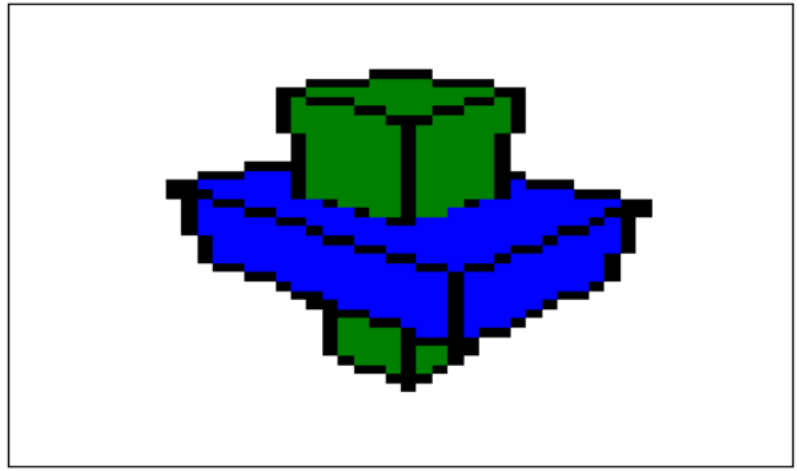


Применение алгоритма Z-буфера к тестовой сцене

Сложность алгоритма зависит от числа пикселей и количества объектов в сцене: $O(c * n)$, где c – число пикселей, n – число объектов.

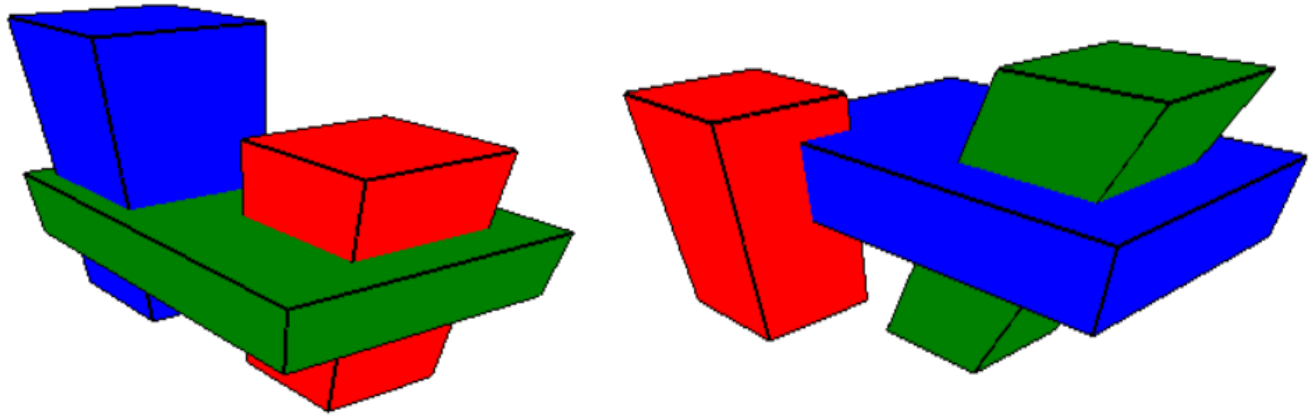


Разрешение экрана 250×250

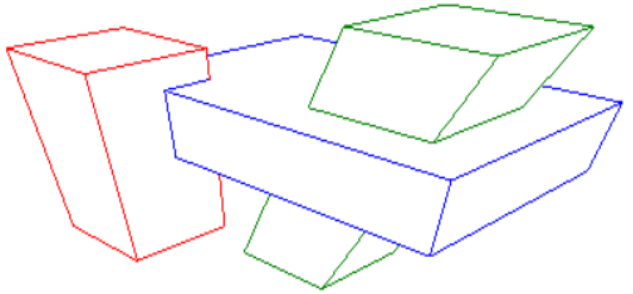


Разрешение экрана 50×50

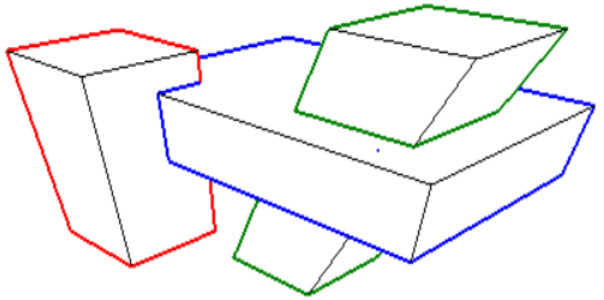
Графический интерфейс



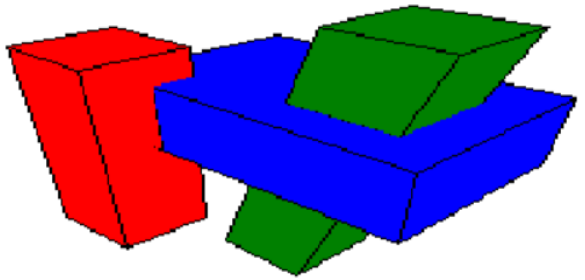
Дополнительные сцены



Алгоритм Робертса

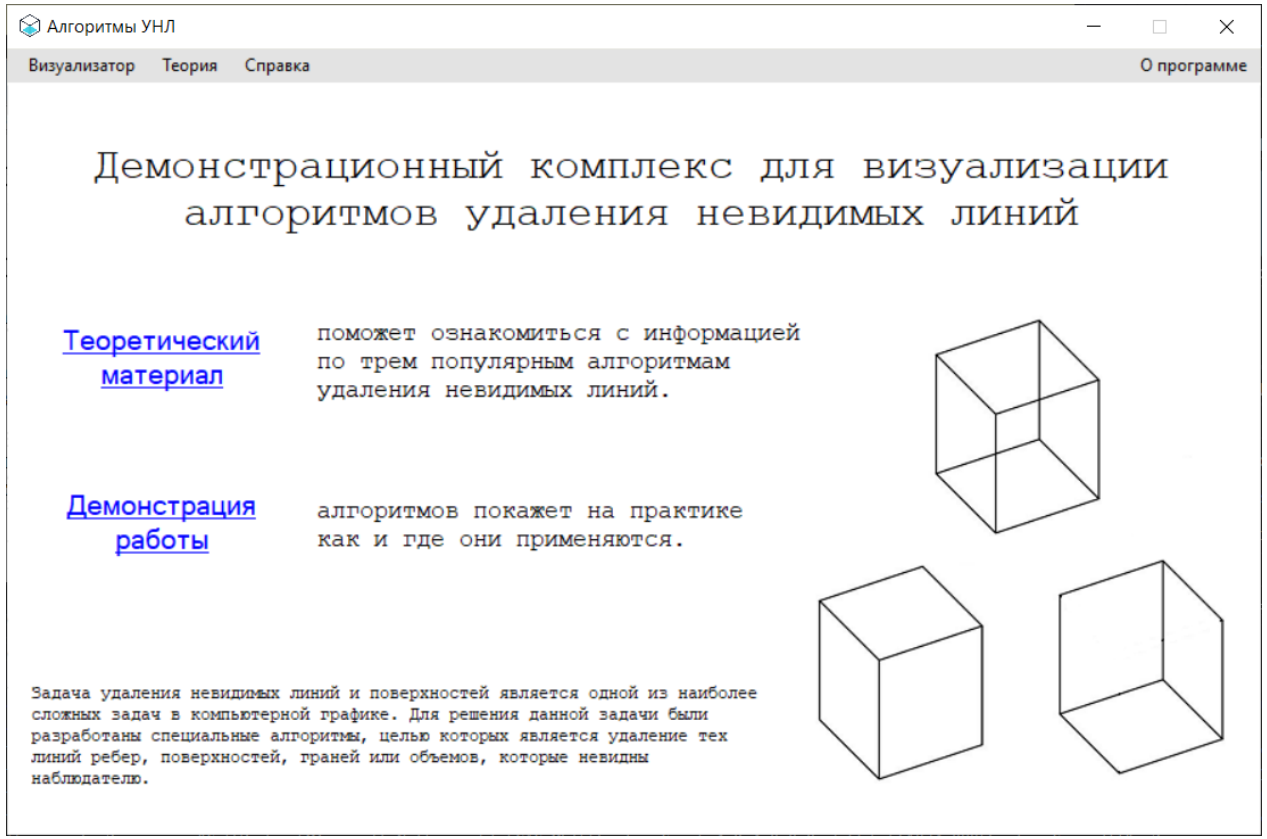


Алгоритм Аппеля

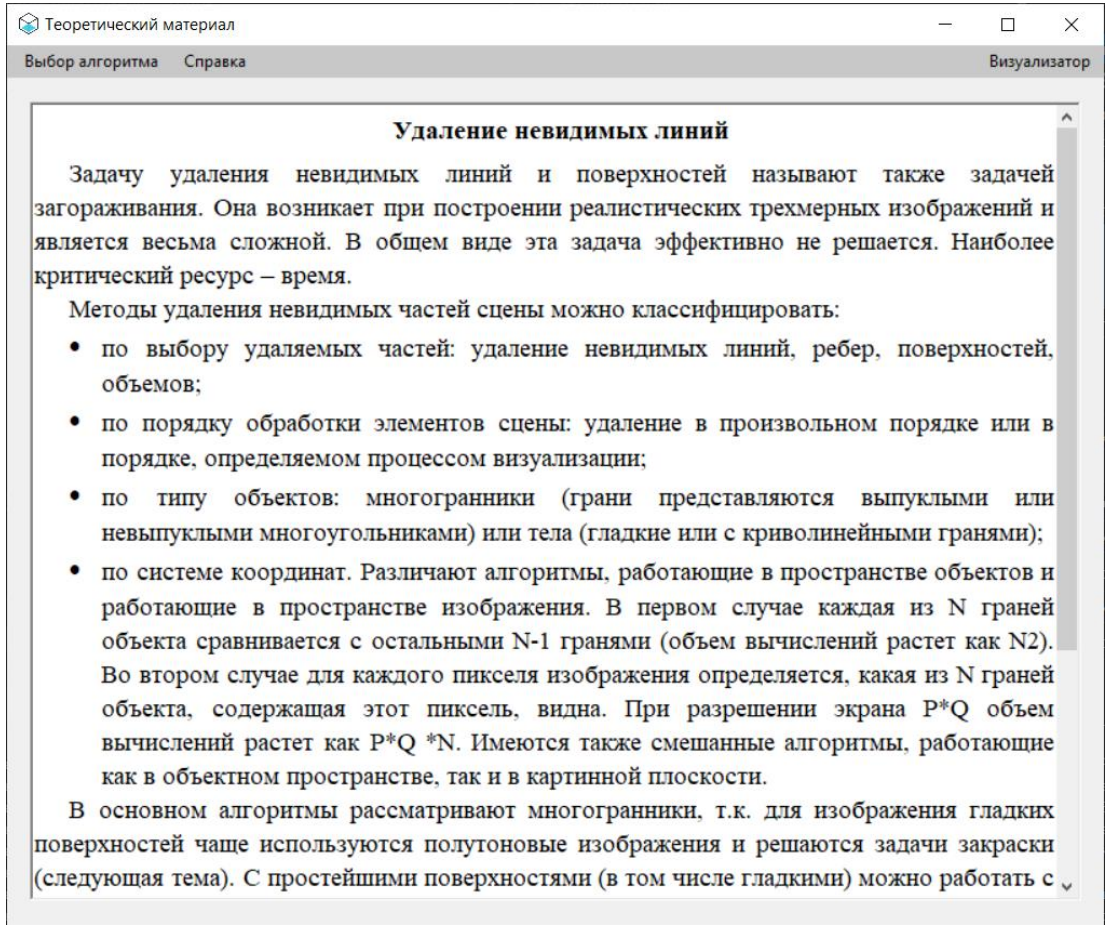


Алгоритм Z-буфера

Графический интерфейс

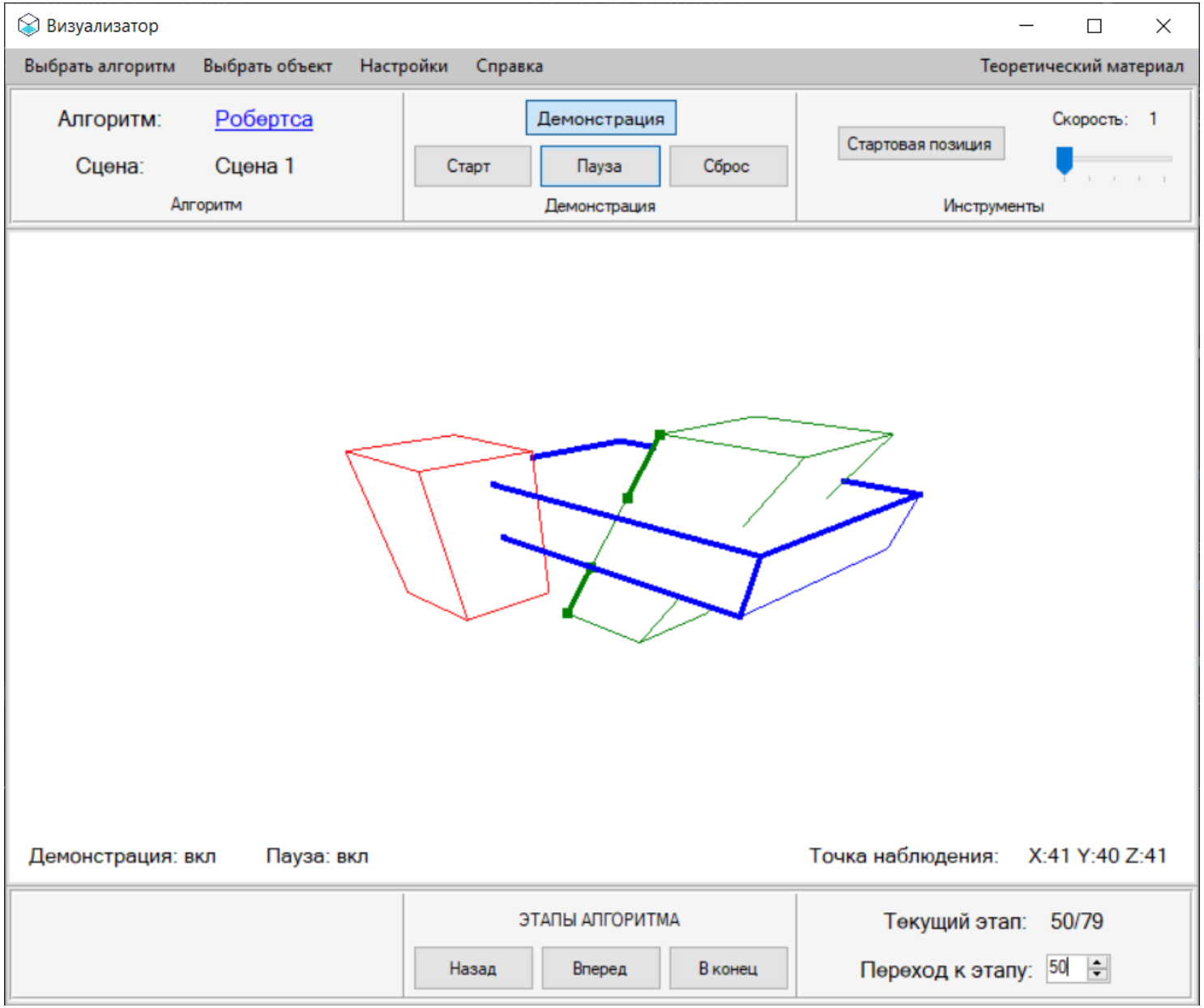


Стартовое меню

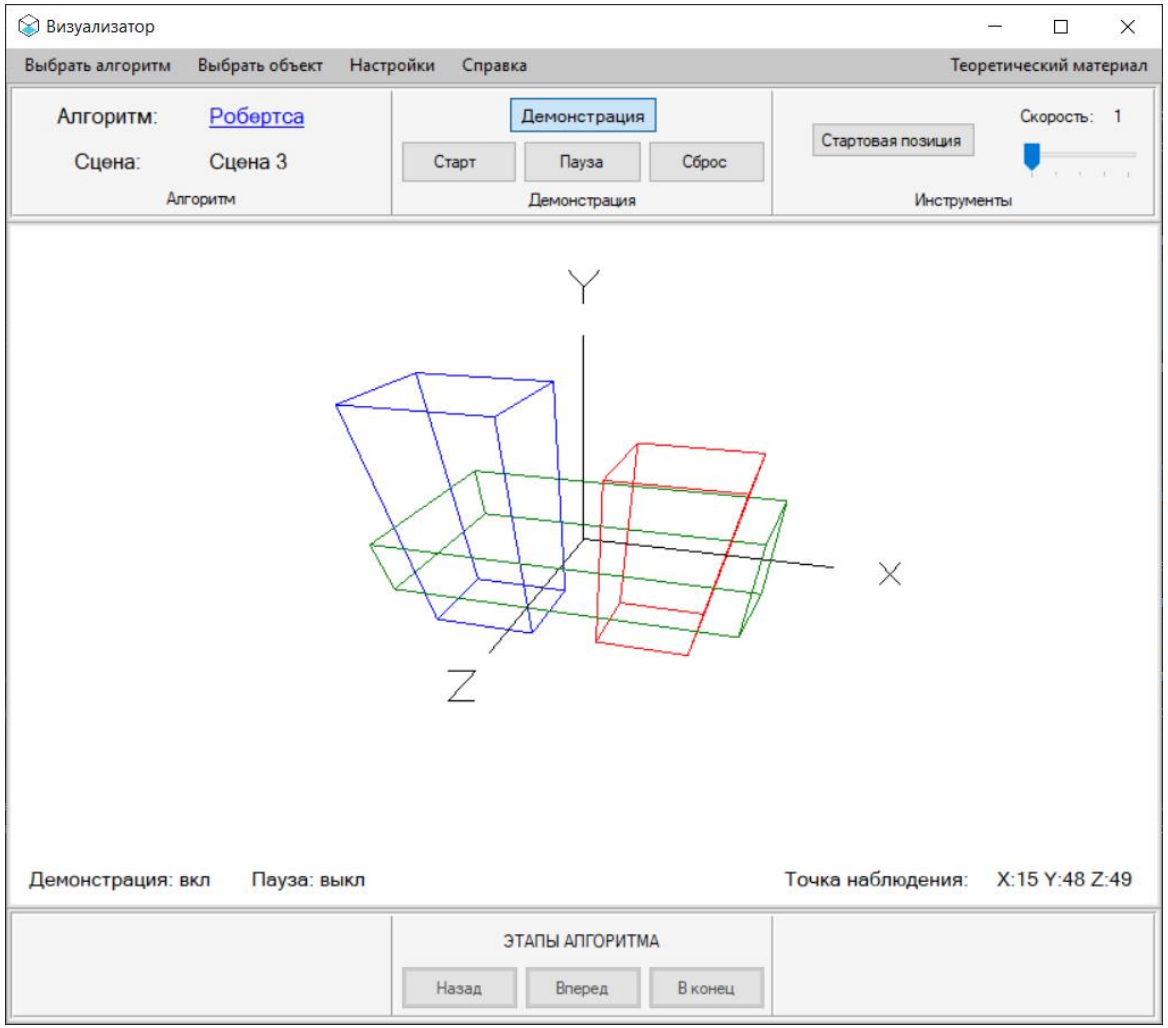


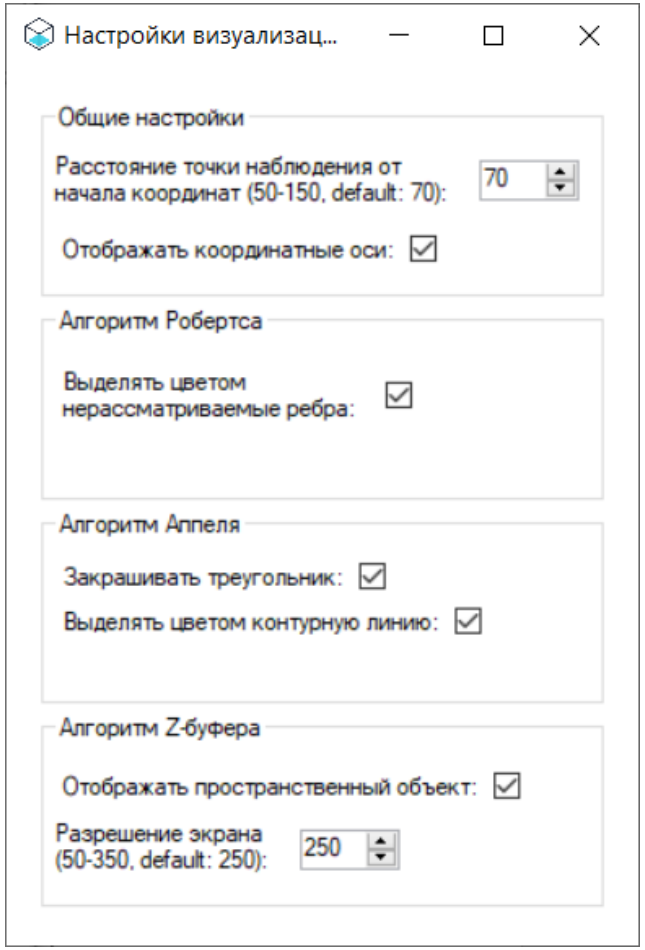
Теория

Графический интерфейс



Графический интерфейс





Основные результаты

Реализованная программная система состоит из 20 классов:

- алгоритмы: Algorithm.cs + 3 класса;
- сцены: SceneToRender.cs + 3 класса;
- графические примитивы: Vertex, Edge, Face, Polyhedron;
- формы:
  - StartMenu.cs;
  - TheoryForm.cs;
  - VisualisatorForm.cs;
  - VisualSettings.cs;
  - Buffers.cs;
- вспомогательные классы: MathMethods.cs, Viewpoint.cs, ChangeCoorPosition.cs.

Основные результаты

В ходе выполнения работы решены следующие задачи:

1. Обоснована актуальность разработки.
2. Проведен обзор аналогов.
3. Выбраны инструменты разработки.
4. Спроектирована архитектура системы.
5. Выполнена программная реализация проекта.

Преимущества разработанного ДПК перед найденными решениями:

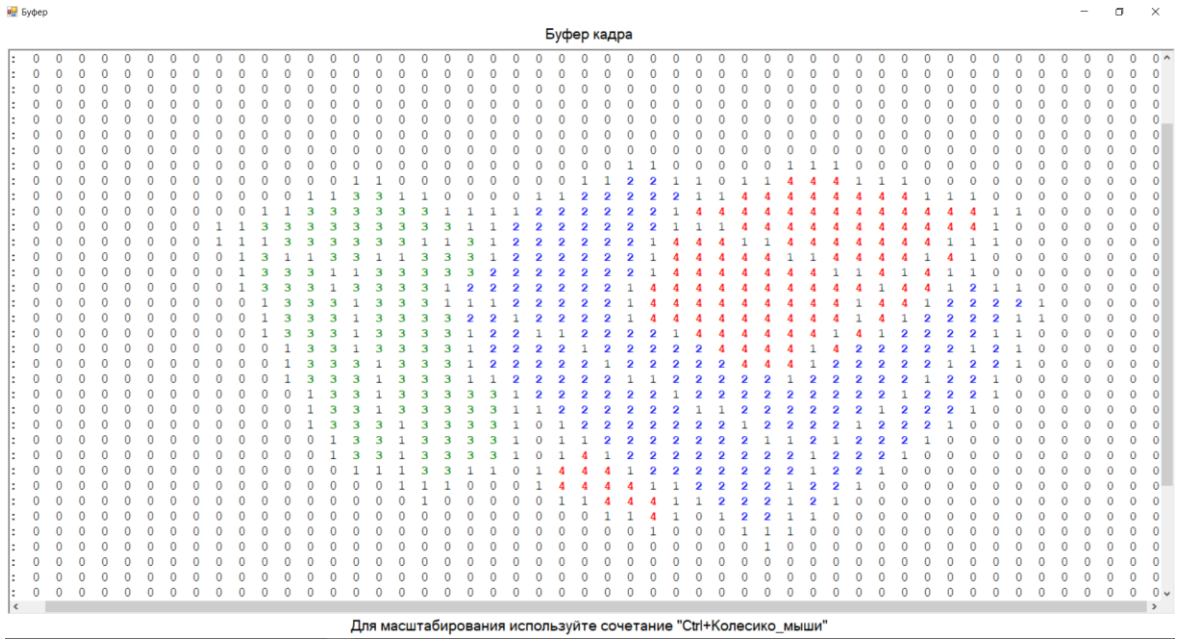
- материал структурирован, сопровождается пояснениями;
- алгоритмы располагаются в одном месте, что обеспечивает удобство и исключает затраты на их поиск;
- детальная демонстрация работы алгоритмов с рассмотрением того, какой объект привел к удалению того или иного ребра;
- возможность производить настройку демонстрации, а также манипулирование ее процессом.

Разработка приложения закончена и передана заказчику.

Полный исходный код ДПК находится в открытом доступе в репозитории GitHub.

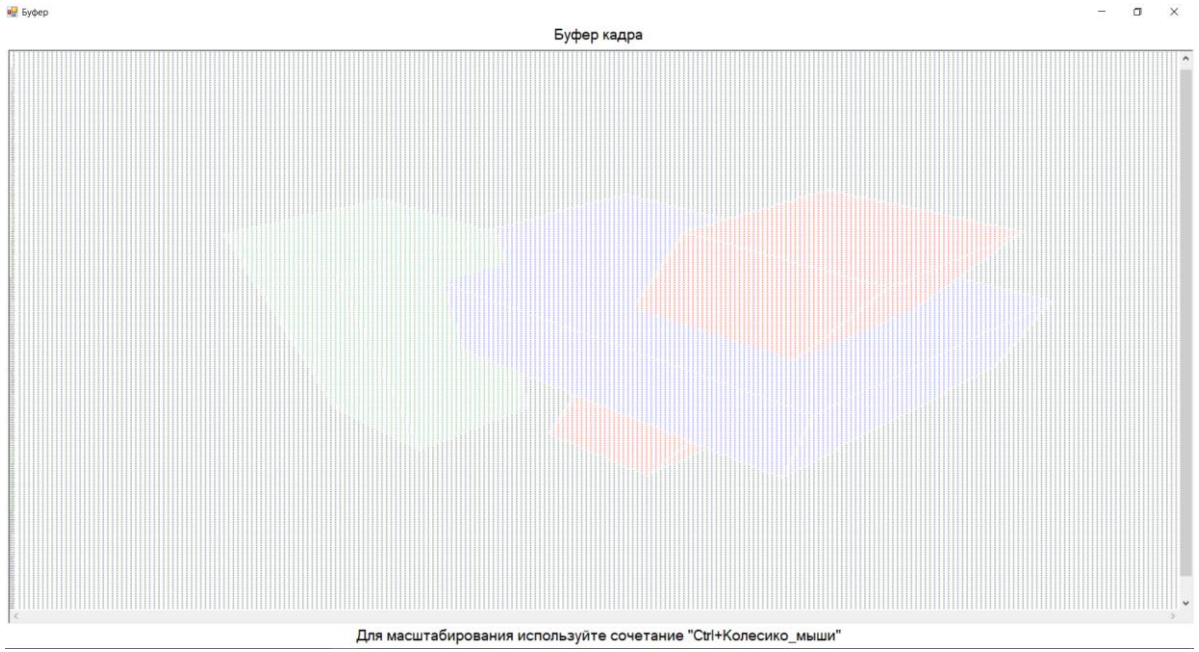
СПАСИБО ЗА ВНИМАНИЕ!

Графический интерфейс

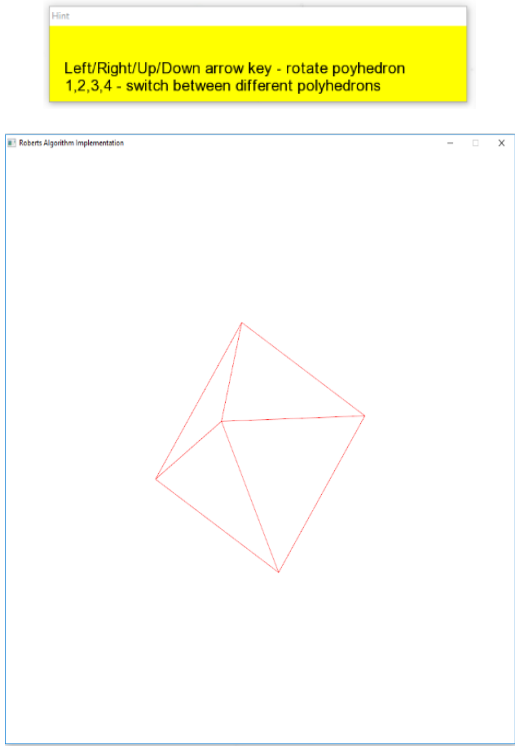


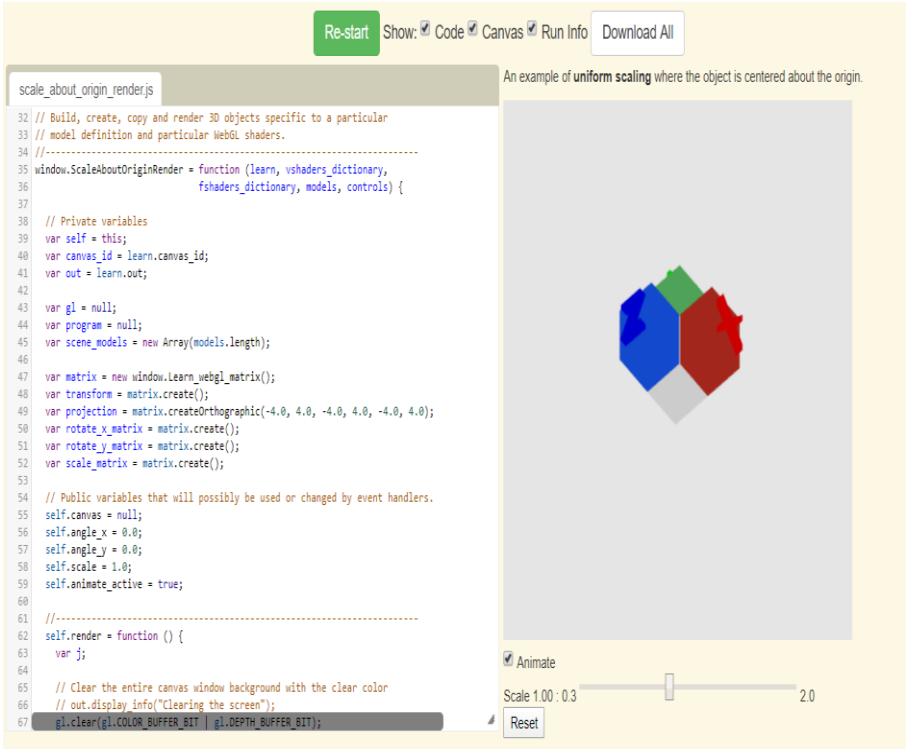
Буфер кадра. Разрешение экрана 50×50

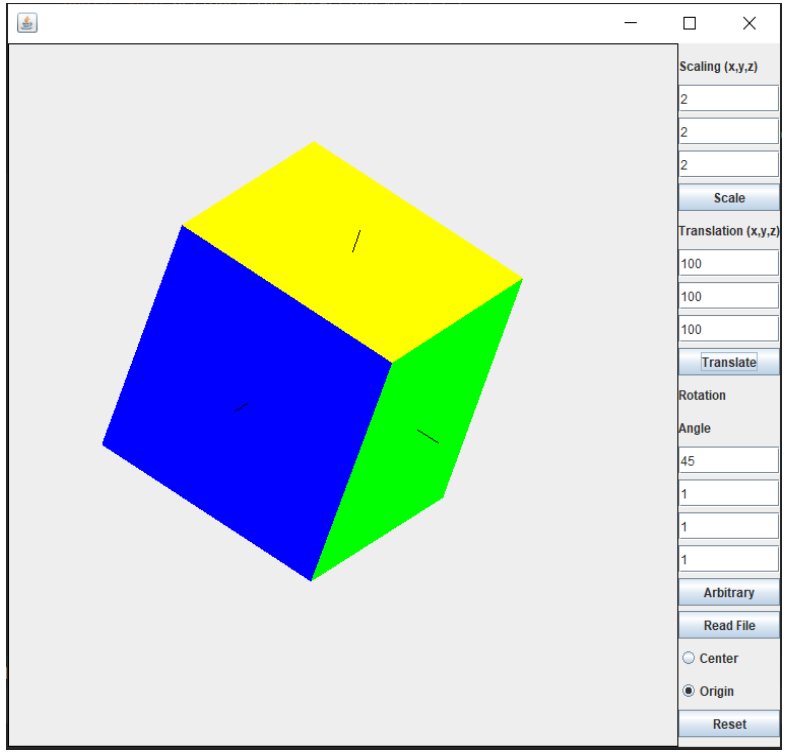
Буфер кадра. Разрешение экрана 250×250



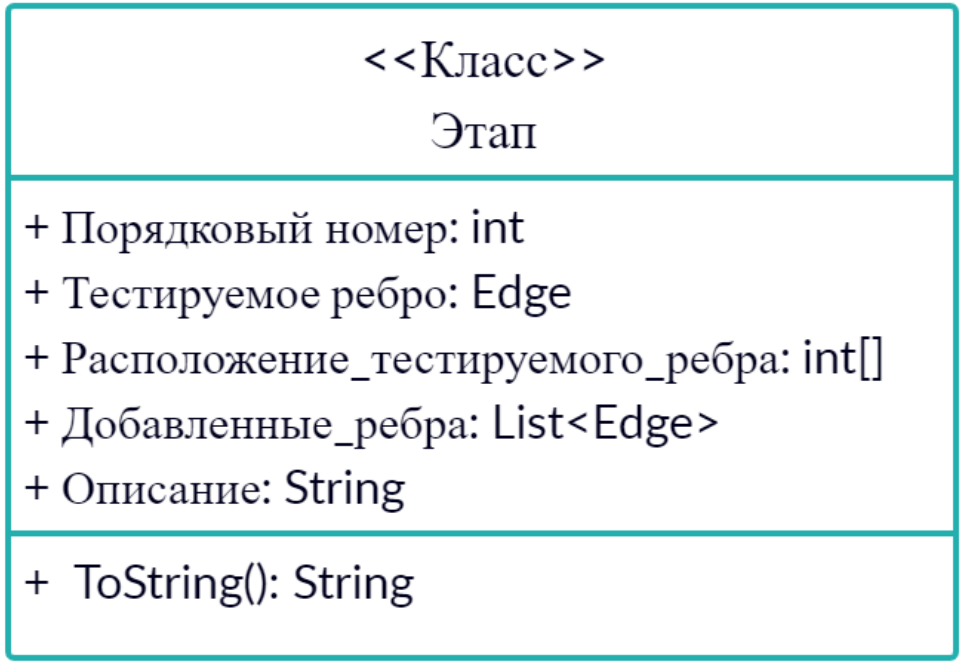
Аналоги

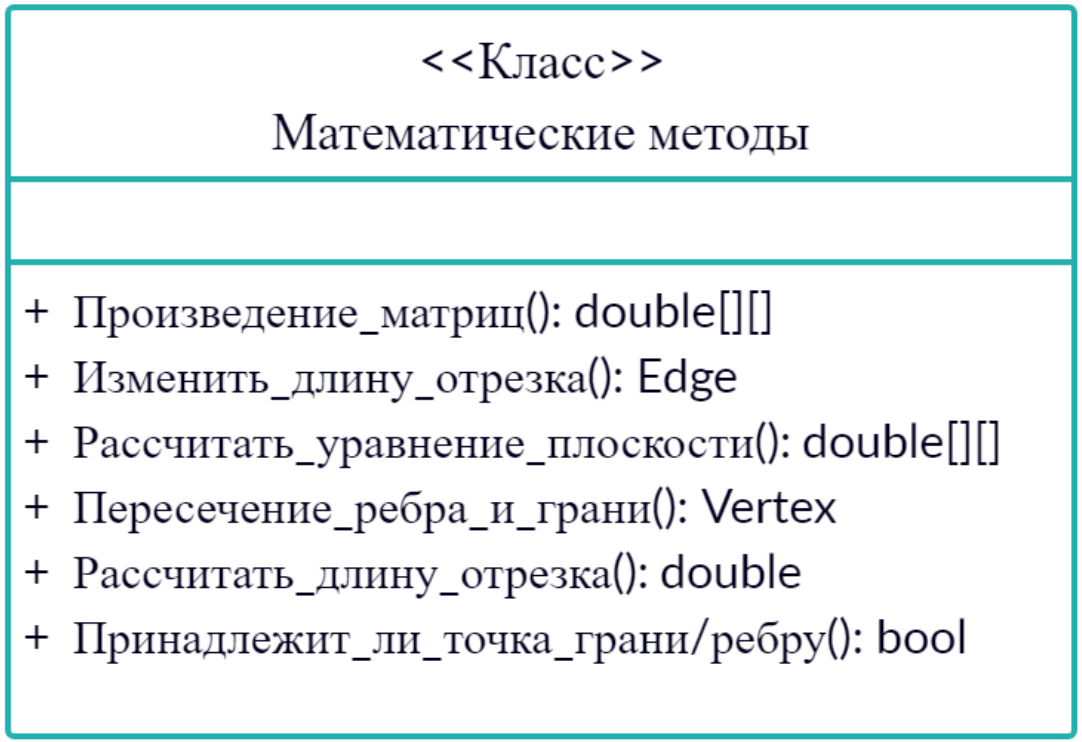

Hint
Left/Right/Up/Down arrow key - rotate poyhedron
1,2,3,4 - switch between different polyhedrons
Roberts Algorithm Implementation


Re-start
Show: Code Canvas Run Info
Download All
scale_about_origin_render.js
An example of uniform scaling where the object is centered about the origin.
// Build, create, copy and render 3D objects specific to a particular
// model definition and particular WebGL shaders.
window.ScaleAboutOriginRender = function (learn, vshaders_dictionary,
fshaders_dictionary, models, controls) {
// Private variables
var self = this;
var canvas_id = learn.canvas_id;
var out = learn.out;
var gl = null;
var program = null;
var scene_models = new Array(models.length);
var matrix = new window.Learn_webgl_matrix();
var transform = matrix.create();
var projection = matrix.createOrthographic(-4.0, 4.0, -4.0, 4.0, -4.0, 4.0);
var rotate_x_matrix = matrix.create();
var rotate_y_matrix = matrix.create();
var scale_matrix = matrix.create();
// Public variables that will possibly be used or changed by event handlers.
self.canvas = null;
self.angle_x = 0.0;
self.angle_y = 0.0;
self.scale = 1.0;
self.animate_active = true;
self.render = function () {
var j;
// Clear the entire canvas window background with the clear color
// out.display_info("Clearing the screen");
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
Animate
Scale 1.00 : 0.3
2.0
Reset

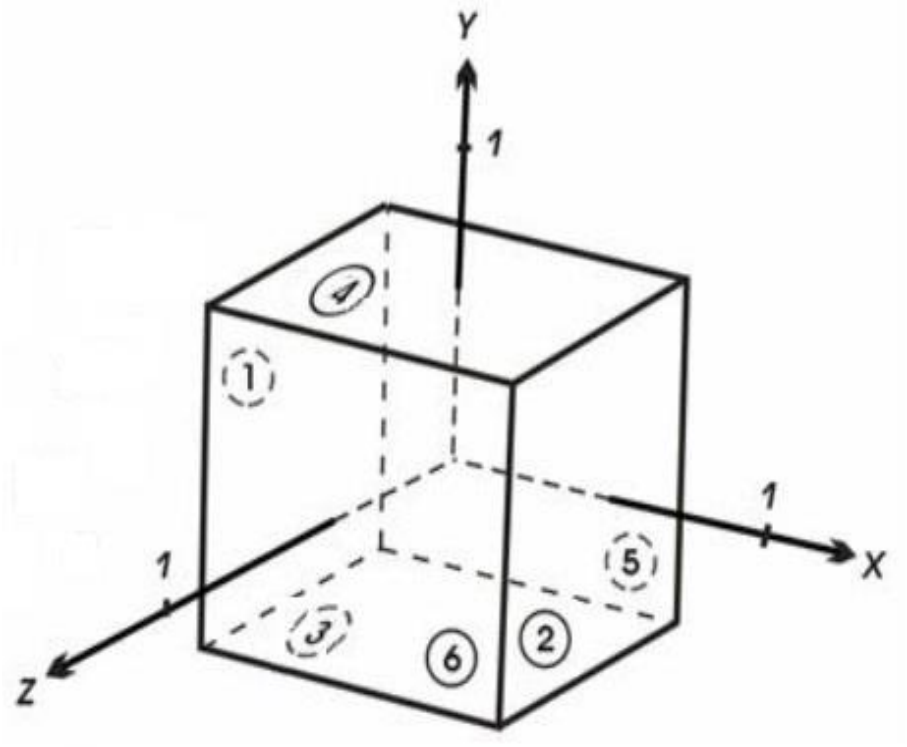

Scaling (x,y,z)
2
2
2
Scale
Translation (x,y,z)
100
100
100
Translate
Rotation
Angle
45
1
1
1
Arbitrary
Read File
Center
Origin
Reset

Вспомогательные классы





Тест видимости. Исходные данные



Единичный куб

Шесть плоскостей, описывающих куб:

$$x_1 = -\frac{1}{2};\ x_2 = \frac{1}{2};\ y_3 = -\frac{1}{2};\ y_4 = \frac{1}{2};\ z_5 = -\frac{1}{2};\ z_6 = \frac{1}{2}.$$

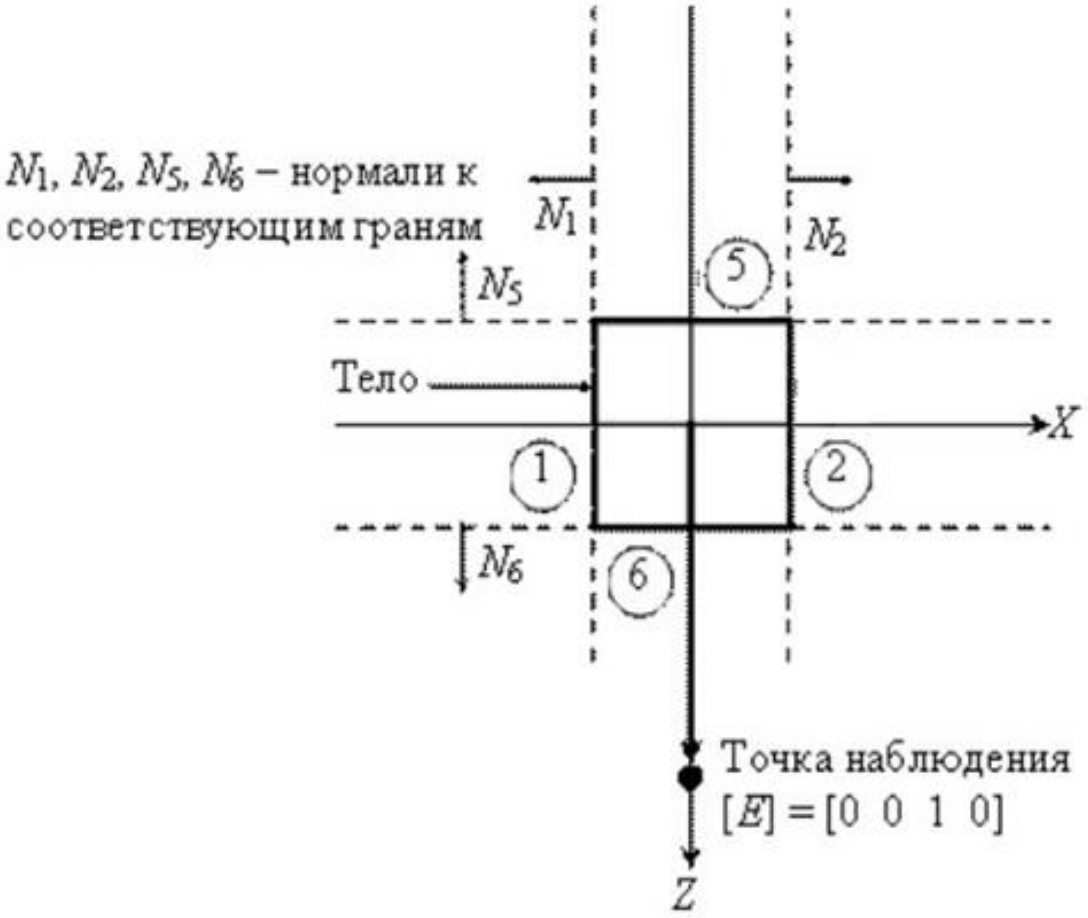
Матрица тела:

$$[V] = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} = \begin{bmatrix} 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 \\ 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}$$

Точка, лежащая внутри куба:

$$x = \frac{1}{4}, y = \frac{1}{4}, z = \frac{1}{4}.$$

Тест видимости. Пример



Матрица тела:

$$[V] = \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

$$[E] * [V] = [0 \quad 0 \quad 1 \quad 0] * \begin{bmatrix} -2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 2 \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

В результате произведения $[E] * [V]$ получается следующая однострочная матрица:

$$[0 \quad 0 \quad 0 \quad 0 \quad -2 \quad 2]$$

Алгоритм Робертса. Определение

Для каждого этапа алгоритма:

P_1P_2 – исследуемое ребро

$\bar{v} = \bar{s} + t * \bar{d}$, где $\bar{s} = P_1$, $\bar{d} = (P_2 - P_1)$

Необходимо найти такое t, при котором изменяется видимость ребра.

$\bar{v} + a * \bar{g}$, где $0 \leq a$

Тогда $\bar{s} + t\bar{d} + a\bar{g}$ – фактически уравнение плоскости.

Необходимо найти такие t и a, для которых:

$\bar{H} = \bar{s} * [V] + t * \bar{d} * [V] + a * \bar{g} * [V] > 0$

Алгоритм Робертса. Пример

Исходные данные:

$P_н = (-15{,}6;\ 8{,}4;\ -10{,}8)$, $P_к = (15{,}6;\ 8{,}4;\ -10{,}8)$ – вершины тестируемого ребра.

$$[V] = \begin{bmatrix} 720 & -720 & 0 & 0 & 0 & 0 \\ 0 & 0 & 720 & -720 & 0 & 0 \\ 0 & 0 & 0 & 0 & 720 & -720 \\ 4320 & 4320 & 5011 & 3628 & 4320 & 4320 \end{bmatrix}$$

$\bar{s} = (-15{,}6;\ 8{,}4;\ -10{,}8); \bar{d} = (31.2;\ 0;\ 0;\ 0);\ \bar{g} = (40;\ 40;\ 40);$

- $\bar{p} = \bar{s} * [V] = (-6912;\ 15552;\ 6220;\ 2419; -3456;\ 12096);$
- $\bar{q} = \bar{d} * [V] = (22464; -22464;\ 0;\ 0;\ 0;\ 0);$
- $\bar{w} = \bar{g} * [V] = (33120;\ -24480; 10771;\ -2131; 33120;\ -24480);$

Система неравенств:

$$h_i = p_i + t * q_i + a * w_i > 0$$

$$\begin{cases} -6912 + t * 22464 + a * 33120 > 0 \\ 15552 + t * -22464 + a * -24480 > 0 \\ 6220{,}8 + t * 0 + a * 10771{,}2 > 0 \\ 2419{,}2 + t * 0 + a * -2131{,}2 > 0 \\ -3456 + t * 0 + a * 33120 > 0 \\ 12096 + t * 0 + a * -24480 > 0 \end{cases}$$

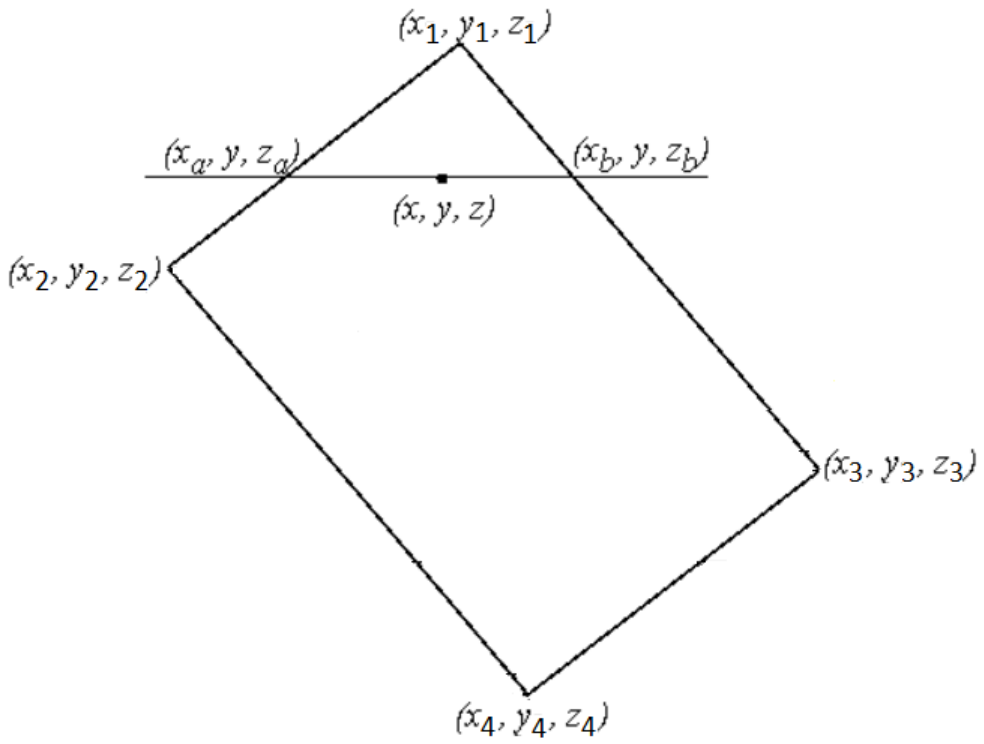
Учитывая ограничения $0 \le a, t \le 1, 0 \le t$:

$t = 0.579$ и $a = 0.104;$

По найденным данным рассчитываются вершины:

$P_2 = P(0.579) = (-15{,}6\ \ 8{,}4\ \ -10{,}8) + (31.2\ 0\ 0\ 0) * 0.579 = (2{,}45;\ 8{,}4;\ -10{,}8);$

Алгоритм Z-буфера. Сканирующая строка



Значение y меняется от y_1 до y_2, от y_2 до y_3, от y_3 до y_4, при этом для каждой строки определяется значения x_a, z_a, x_b, z_b:

$$x_a = x_1 - (x_2 - x_1) * \frac{y - y_1}{y_2 - y_1};$$

$$x_b = x_1 - (x_2 - x_1) * \frac{y - y_1}{y_3 - y_1};$$

$$z_a = x_1 - (z_2 - z_1) * \frac{y - y_1}{y_2 - y_1};$$

$$z_b = z_1 - (z_3 - z_1) * \frac{y - y_1}{y_3 - y_1};$$

На сканирующей строке x меняется от x_a до x_b и для каждой точки строки определяется глубина z:

$$z = z_a - (z_b - z_a) * \frac{x - x_a}{x_b - x_a};$$