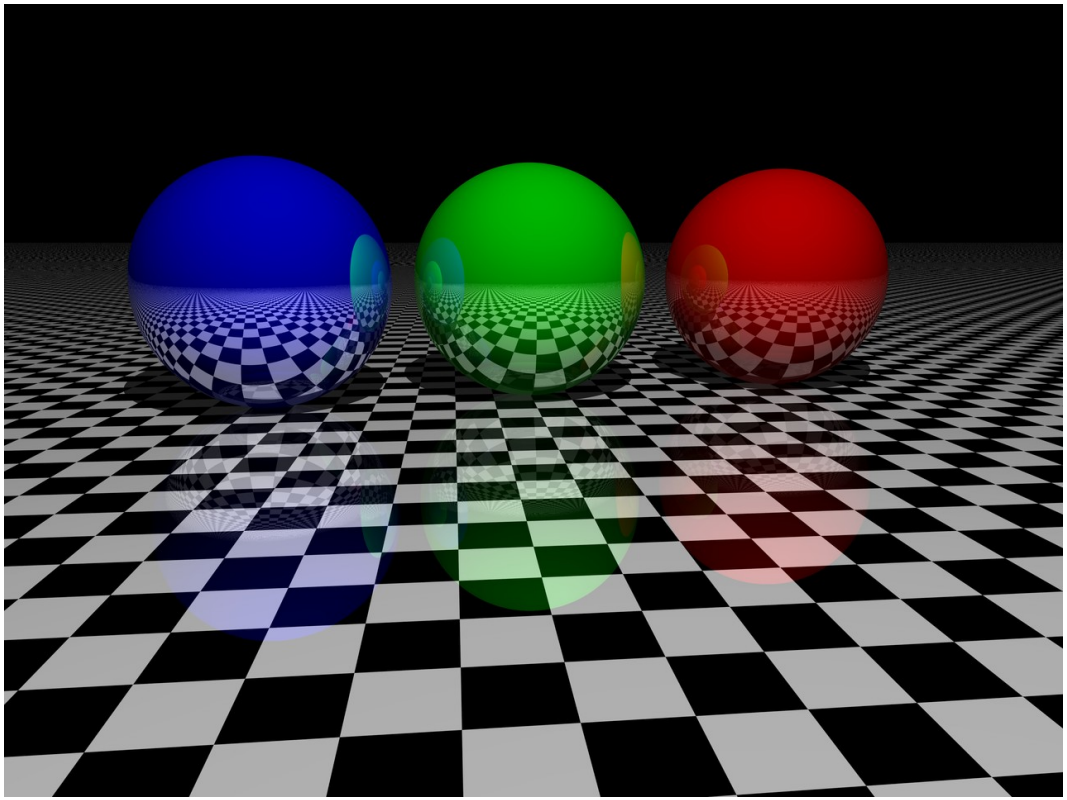
Основы трехмерной графики

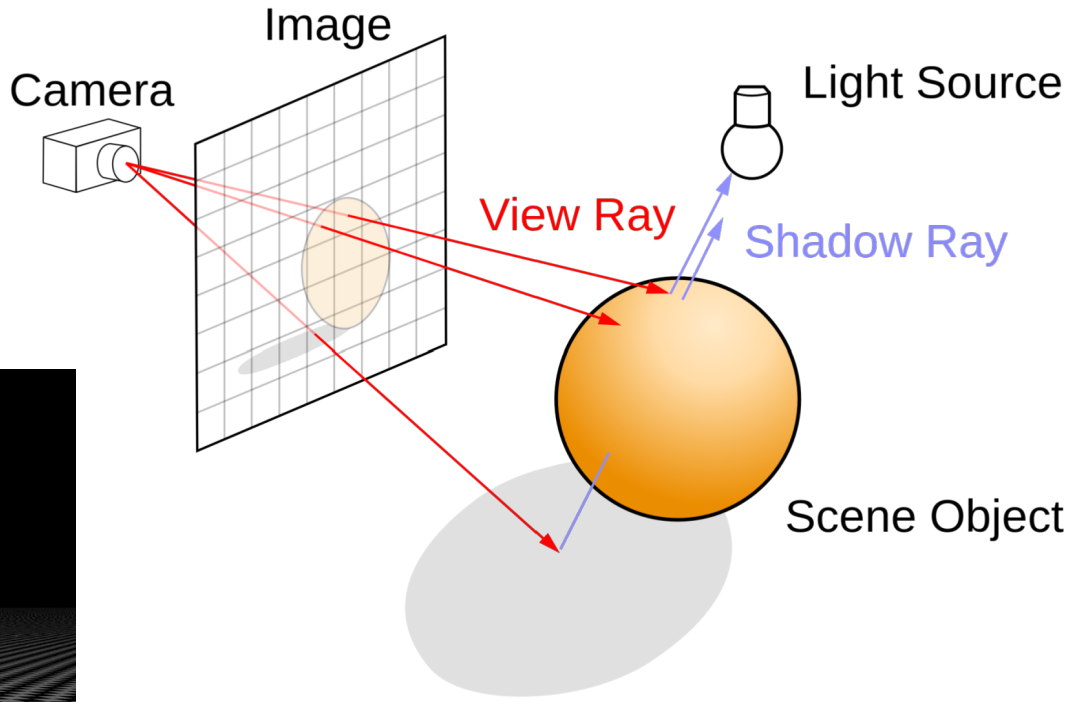
Необходимо сформировать проекцию трехмерного объекта или сцены (набора объектов) на двумерный экран.

- Метод трассировки лучей. Телесный угол луча равен телесному углу пикселя.
- Метод аппроксимации объектов конечным количеством геометрических примитивов: плоскости, криволинейные поверхности.
  - При применении плоскостей для аппроксимации объектов позволяет рассчитывать их изображение в реальном масштабе времени.
  - Применение специальных вычислителей позволяет повысить детализацию и снизить грубость аппроксимации.

Трассировка лучей

- Позволяет точно моделировать отражение и преломление
- Подходит для фотореалистичной графики





- Огромные вычислительные затраты
- Не подходит для работы в режиме реального времени

Полигональная графика

Трехмерный объект описывается набором характерных точек — вершин. Вершины соединены ребрами, формирующими плоскости или **полигоны**.

На каждый полигон наносится двумерное изображение — **текстура**. Как правило, используется текстура-развертка, а каждой вершине соответствует определенные двумерные координаты в этой текстуре.

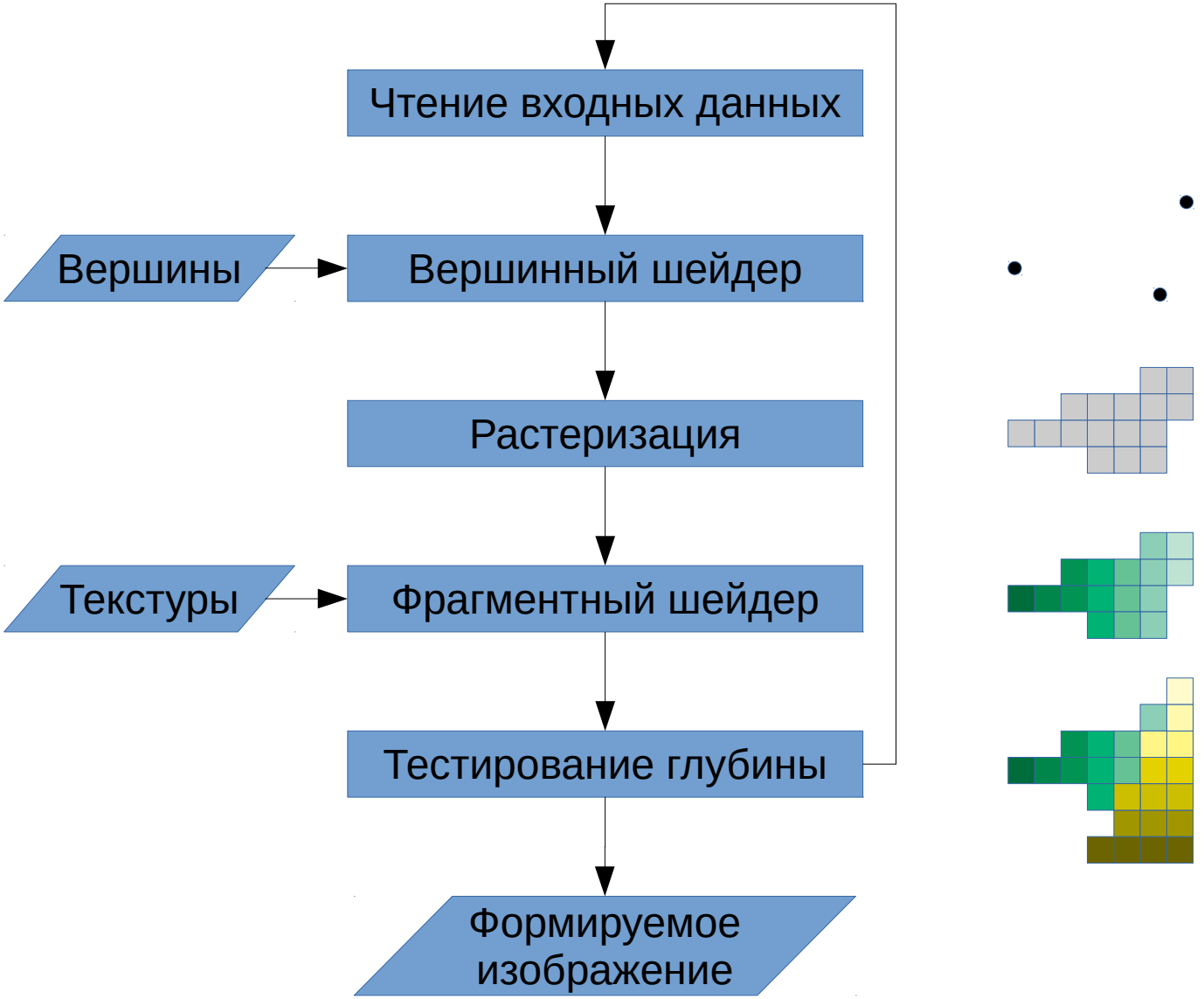
Каждая вершина объекта или точка текстуры может быть спроецирована на экран методами линейной алгебры.

Рассчитывается освещение и корректируется цвет текстуры.

Полигоны отрисовываются с учетом перекрытия друг друга: более близкие полигоны рисуются поверх более далёких, но не наоборот.

Полигональная графика

Графический конвейер



Графический конвейер

Современная трехмерная графика характеризуется количеством вершин объектов порядка сотен тысяч и количеством точек текстур порядка миллионов.

Вычислительно, наиболее сложная задача — расчет текстурирования, так как она включает в себя расчет большого числа точек. С другой стороны, операции, выполняемые над всеми точками идентичны, различаются лишь входные данные.

Это позволило разработать специальное аппаратное обеспечение — графический ускоритель (GPU, Graphics Processing Unit). Он содержит огромное количество простых вычислительных ядер, умеющих выполнять одинаковую операцию над разными данными одновременно.

Графический конвейер

Шейдер — программа, исполняемая на GPU и выполняющая расчет графической сцены.

Различают **фиксированный** графический конвейер, в котором последовательность и набор операций определены производителем GPU, и **программируемый** графический конвейер, позволяющий исполнять на GPU произвольный код в рамках шейдера.

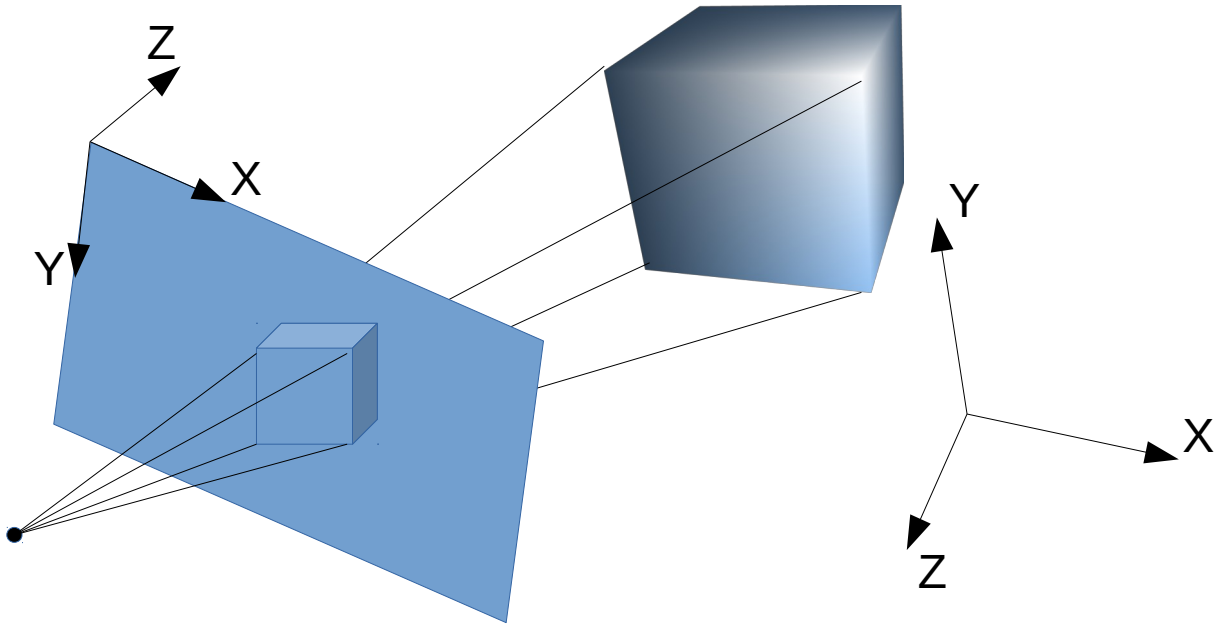
Современные GPU предоставляют программируемый графический конвейер и даже позволяют полностью отказаться от него для специальных задач рендеринга и высокопроизводительных вычислений.

Вершинный шейдер

Задача вершинного шейдера — пересчитать координаты вершин из трехмерных координат, в которых осуществляется моделирование, в двумерные координаты экрана.

При расчете проецирования используется матрица однородного преобразования 4x4.

$$T=\begin{pmatrix} R & \mathrm{P} \\ f & m \end{pmatrix}=\begin{pmatrix} r_{11} & r_{12} & r_{13} & \rho_1 \\ r_{21} & r_{22} & r_{23} & \rho_2 \\ r_{31} & r_{32} & r_{33} & \rho_3 \\ f_1 & f_2 & f_3 & m \end{pmatrix}$$



Фрагментный шейдер

Фрагментный шейдер рассчитывает цвет конкретного пикселя, исходя из его параметров:

- Координаты в трехмерном пространстве
- Цвет пикселя текстуры для данного пикселя
- Координаты источников освещения
- Координаты вектора нормали к поверхности

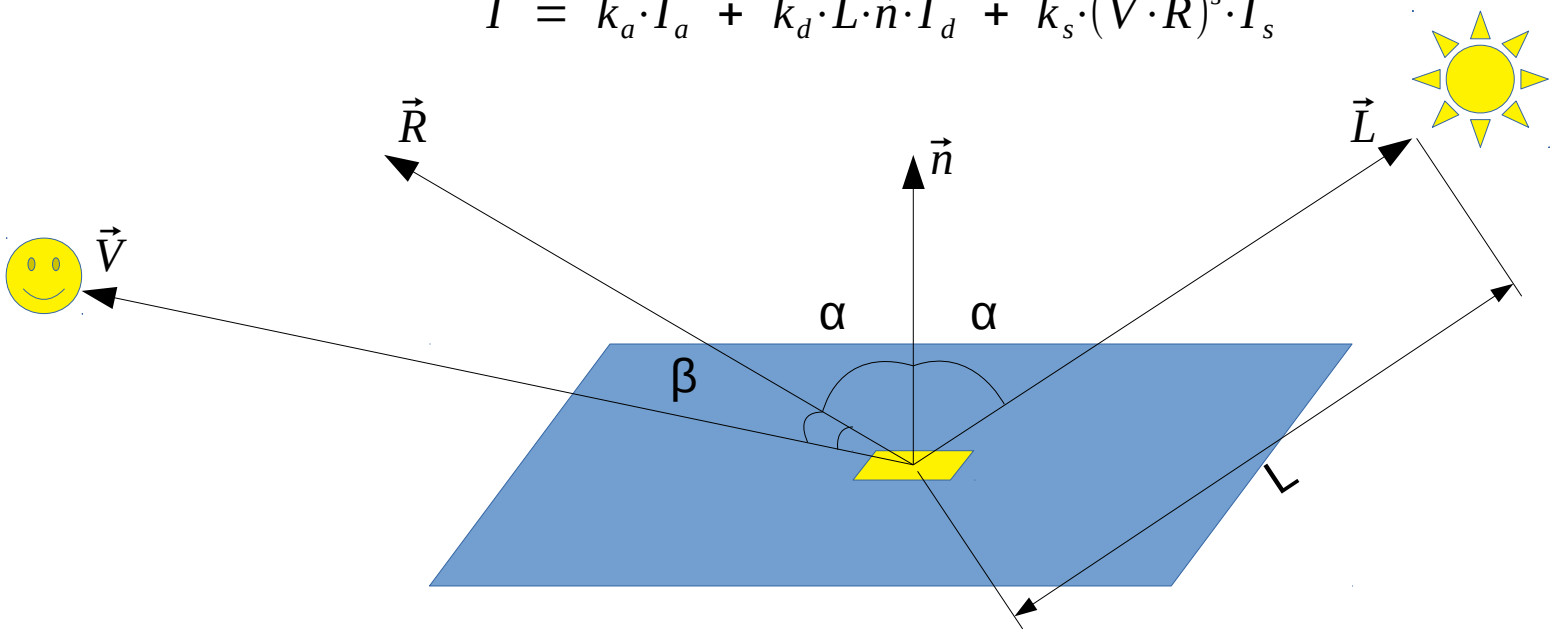
Фиксированный конвейер рассчитывает цвет и освещенность исходя из **модели фонга**.

Программируемый конвейер может свободно оперировать предоставленными данными для расчета цвета пикселя. Также в рамках программируемого конвейера можно предоставлять дополнительные данные.

Модель освещения по Фонгу

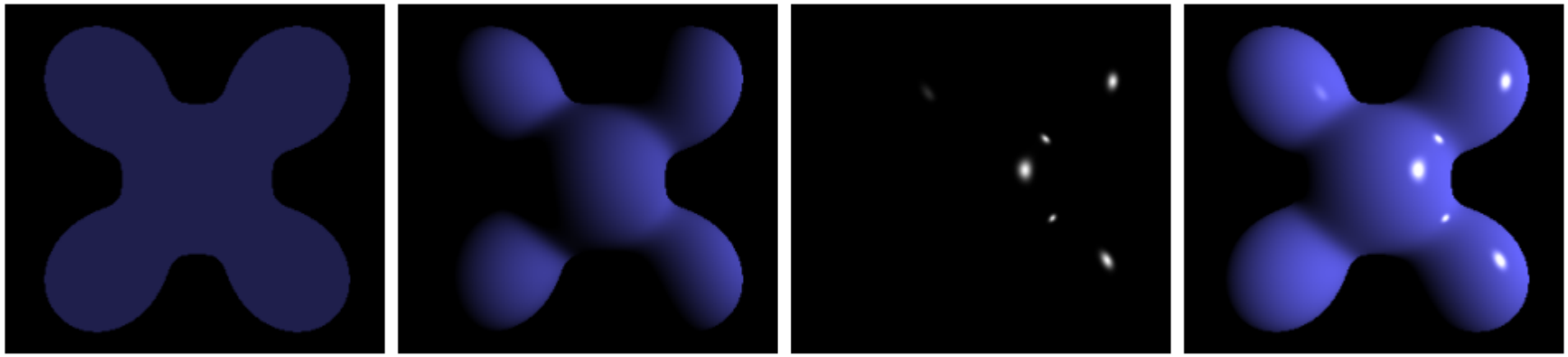
- **Фоновое освещение** (ambient). Зависит от расстояния до источника света.
- **Рассеянное освещение** (diffuse). Зависит от расстояния и угла падения.
- **Зеркальное освещение** (specular). Зависит от расстояния, угла падения и угла наблюдения.

$$I = k_a \cdot I_a + k_d \cdot \vec{L} \cdot \vec{n} \cdot I_d + k_s \cdot (\vec{V} \cdot \vec{R})^s \cdot I_s$$



Модель освещения по Фонгу

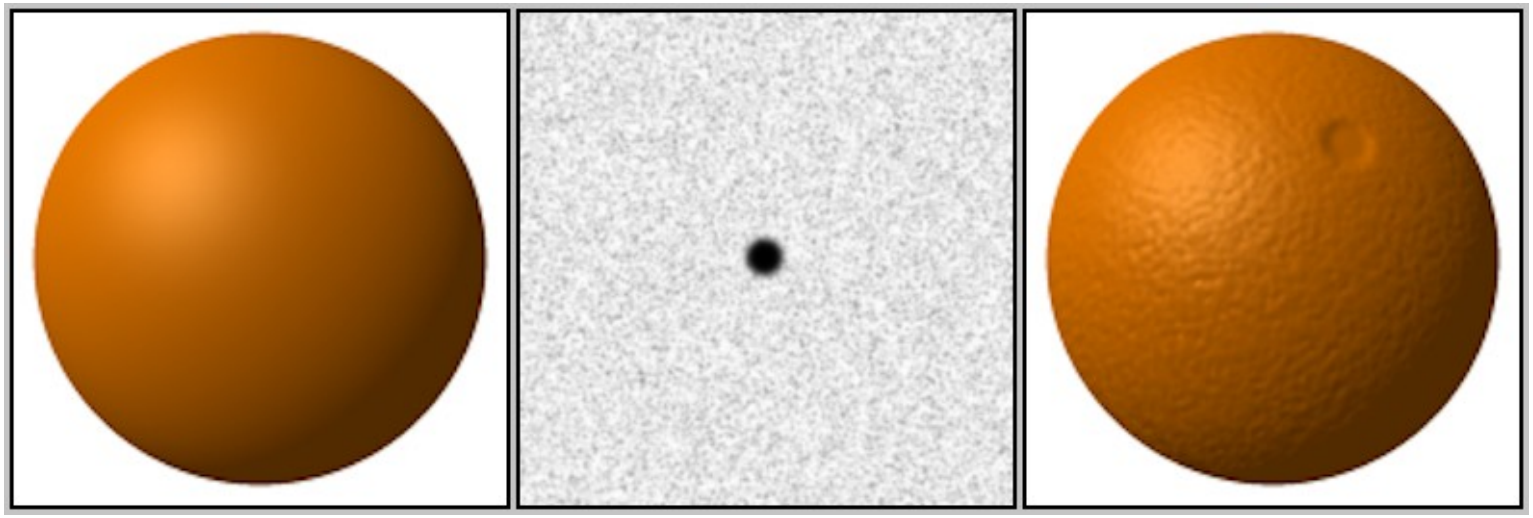
- **Фоновое освещение** (ambient). Зависит от расстояния до источника света.
- **Рассеянное освещение** (diffuse). Зависит от расстояния и угла падения.
- **Зеркальное освещение** (specular). Зависит от расстояния, угла падения и угла наблюдения.



Bump Mapping

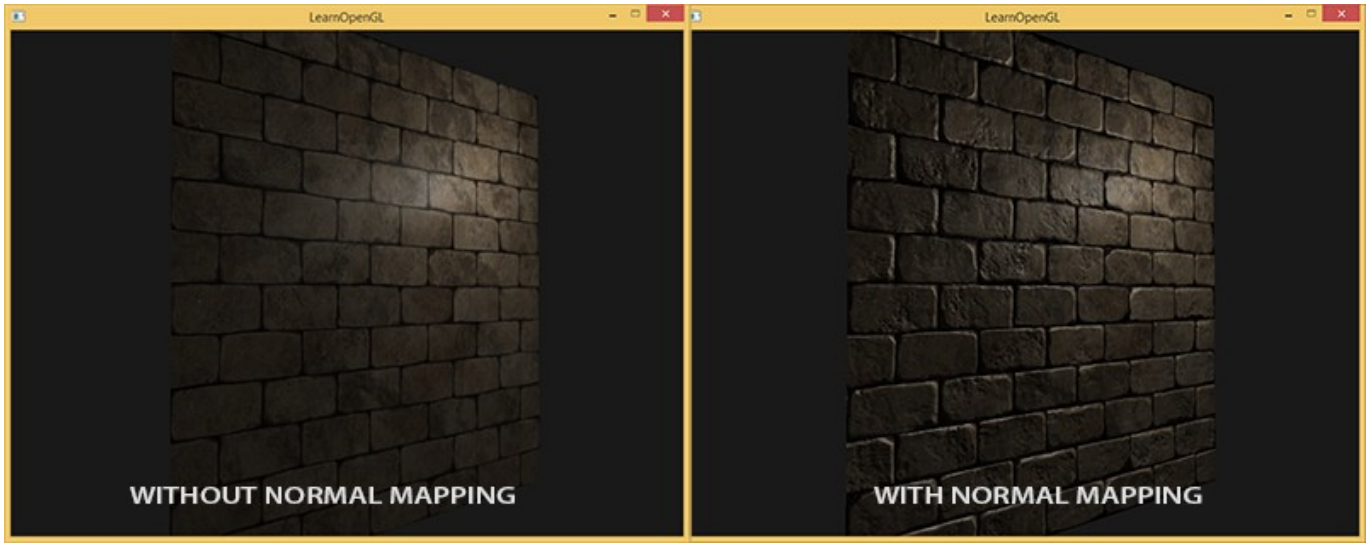
При задании данных нормаль определяется отдельно для каждой вершины. Нормаль конкретного пикселя вычисляется интерполяцией нормалей вершин.

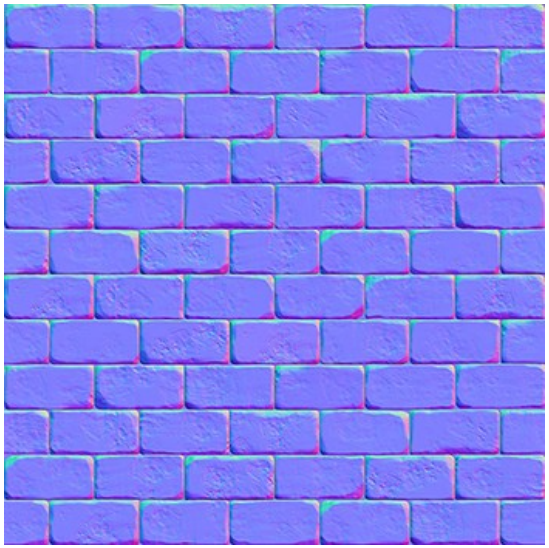
Bump Mapping — это технология добавления карты высоты к текстуре, позволяющая изменить нормаль в каждой точке полигона. Карта bump map — это черно-белая текстура, в которой интенсивность цвета определяет смещение точки относительно поверхности.



Normal Mapping

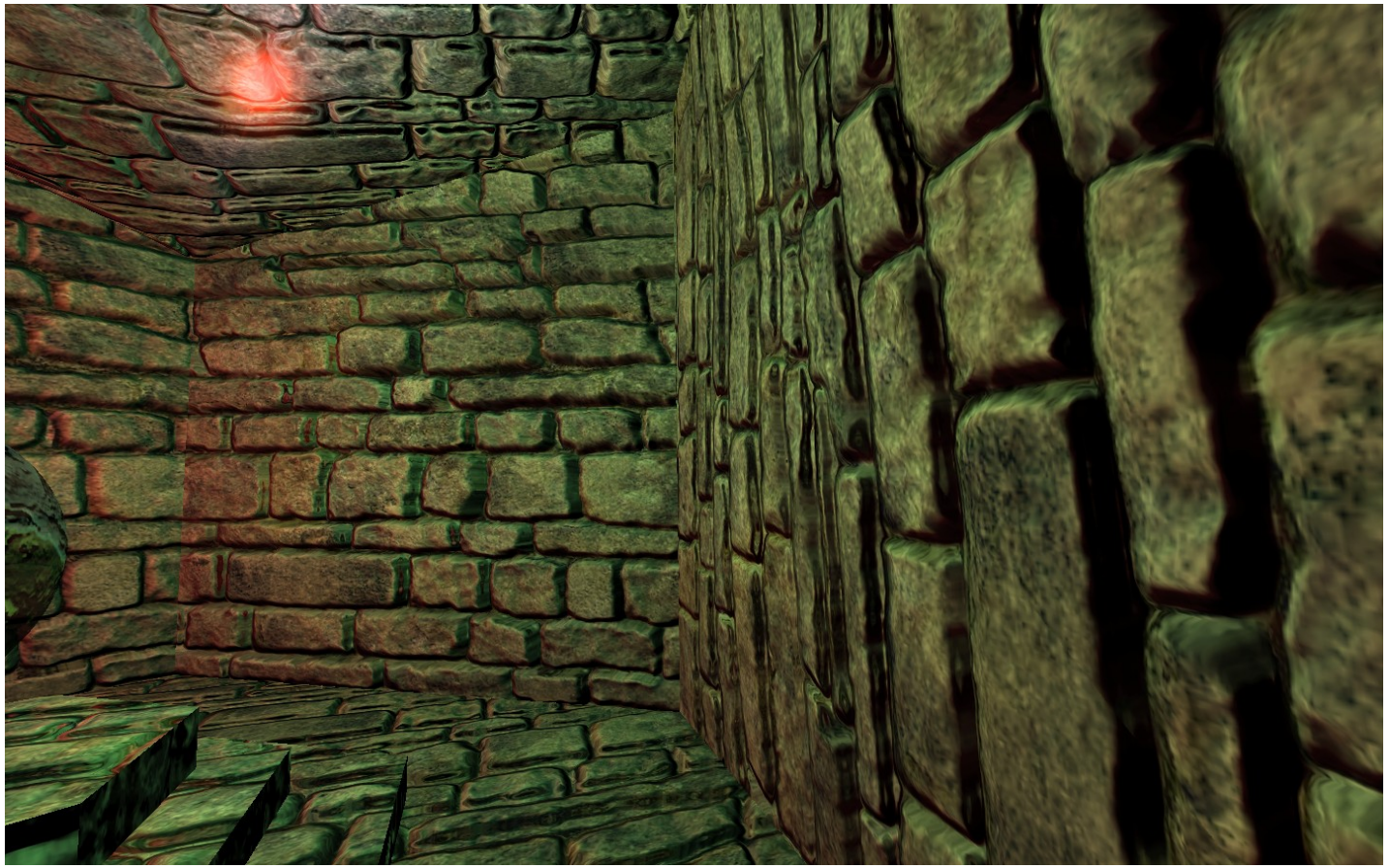
Normal Mapping позволяет задать поправку к нормали в виде трехмерного вектора. Карта нормали — это цветная текстура, составляющие цвета которой — это координаты вектора нормали в каждой точке.





Parallax Mapping

Parallax Mapping позволяет одновременно задать смещение точки поверхности и изменение ее нормали для создания эффекта рельефной поверхности на уровне фрагментного шейдера.



OpenGL

Стандарт разработки приложений с трехмерной графикой.

Описывает требования к аппаратному и программному обеспечению.

Первый стандарт появился в 1994 году.

Стандарт OpenGL 2.0 (2001 г) ввел поддержку языка GLSL для программирования шейдеров. Фиксированный конвейер поддерживает только простую модель освещения по Фонгу.

Последний стандарт OpenGL 4.6 поддерживается всем современным оборудованием.

Существуют реализации OpenGL для большинства современных платформ, включая OpenGL ES (для встраиваемых систем) и WebGL (для веб страниц).

Поддерживаются Compute Shaders для проведения вычислений общего назначения на GPU.

OpenGL

Стандарт описывает в том числе требования к программному коду, использующему OpenGL.

На уровне стандарта определены типы данных, универсальные для всех платформ.

Тип OpenGL	Тип C	Описание
GLenum	unsigned int	Перечислимый тип
GLvoid	void	
GLbyte	signed char	1 байт, целое знаковое
GLshort	short	2 байта, целое знаковое
GLint	int	4 байта, целое знаковое
GLubyte	unsigned char	1 байт, целое беззнаковое
GLushort	unsigned short	2 байта, целое беззнаковое
GLuint	unsigned int	4 байта, целое беззнаковое
GLsizei	int	4 байта, знаковое
GLfloat	float	Вещественное число одинарной точности
GLdouble	double	Вещественное число двойной точности

OpenGL

Константы OpenGL объявлены заглавными буквами.

```
GL_COLOR_BUFFER_BIT
```

Функции OpenGL имеют префикс gl и суффикс, определяющий количество и тип параметров.

```
glVertex2f(GLfloat x, GLfloat y)
glVertex3i(GLint x, GLint y, GLint z)
```

Реализация OpenGL возложена на библиотеку gl (либо opengl32.dll в среде Win32).

Помимо базового функционала, доступны библиотеки:

- GLU (OpenGL Utility) — вспомогательные функции и рисование геометрических примитивов.
- GLUT (OpenGL Utility Toolkit) — функции для создания окна и обработки ввода-вывода.

OpenGL

Перед началом отрисовки необходимо задать размер окна вывода. Функция определяет размер буфера памяти для сохранения конечного изображения.

```
glViewport(0, 0, width, height)
```

Также необходимо включить необходимый функционал OpenGL. Для рисования трехмерных изображений необходимо включить буфер глубины.

```
glEnable(GL_DEPTH_TEST)
```

Необходимо также определить цвет, которым будет закрашиваться буфер при очистке.

```
glClearColor(0, 0, 0, 0)
```

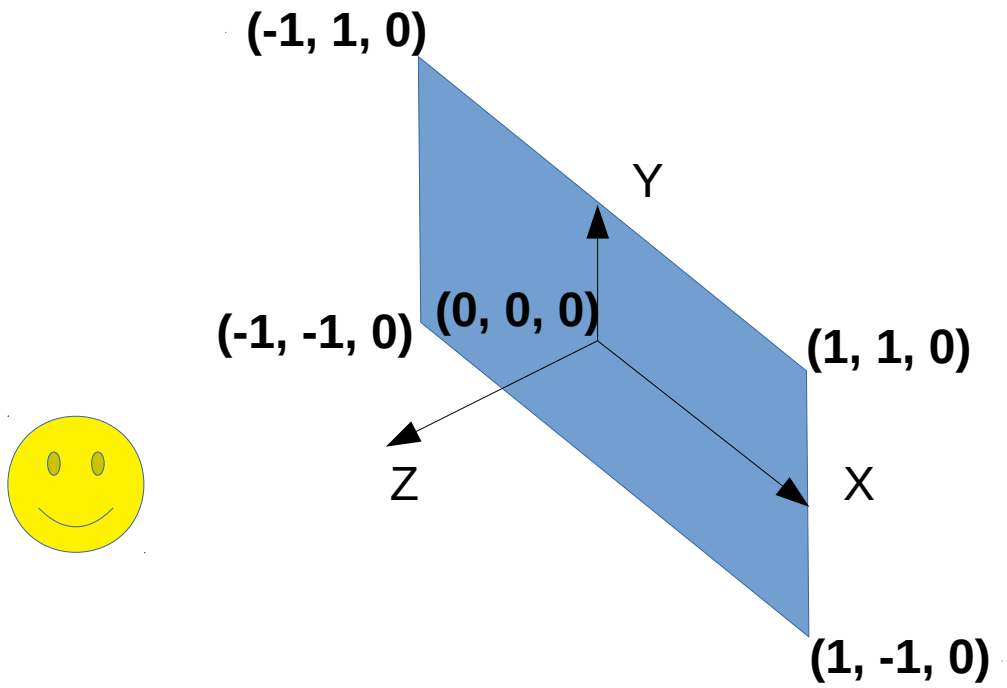
Перед началом рисования следует очистить выходной буфер.

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
```

Матрицы в OpenGL

OpenGL использует последовательно две матрицы однородного преобразования 4x4 для перевода координат вершин в экранные координаты: **матрица проецирования** и **матрица моделирования**. Изначально обе матрицы — единичные.

Единичной матрице преобразования соответствует следующее расположение базиса относительно экрана:



Матрицы в OpenGL

Для выбора текущей матрицы используется функция **glMatrixMode**:

```
glMatrixMode(GL_PROJECTION)  // Матрица проецирования
glMatrixMode(GL_MODELVIEW)   // Матрица моделирования
```

Дальнейшие геометрические преобразования применяются к текущей выбранной матрице путем ее умножения на матрицу преобразования.

Сброс текущей матрицы в единичную производится путем вызова функции **glLoadIdentity()**.

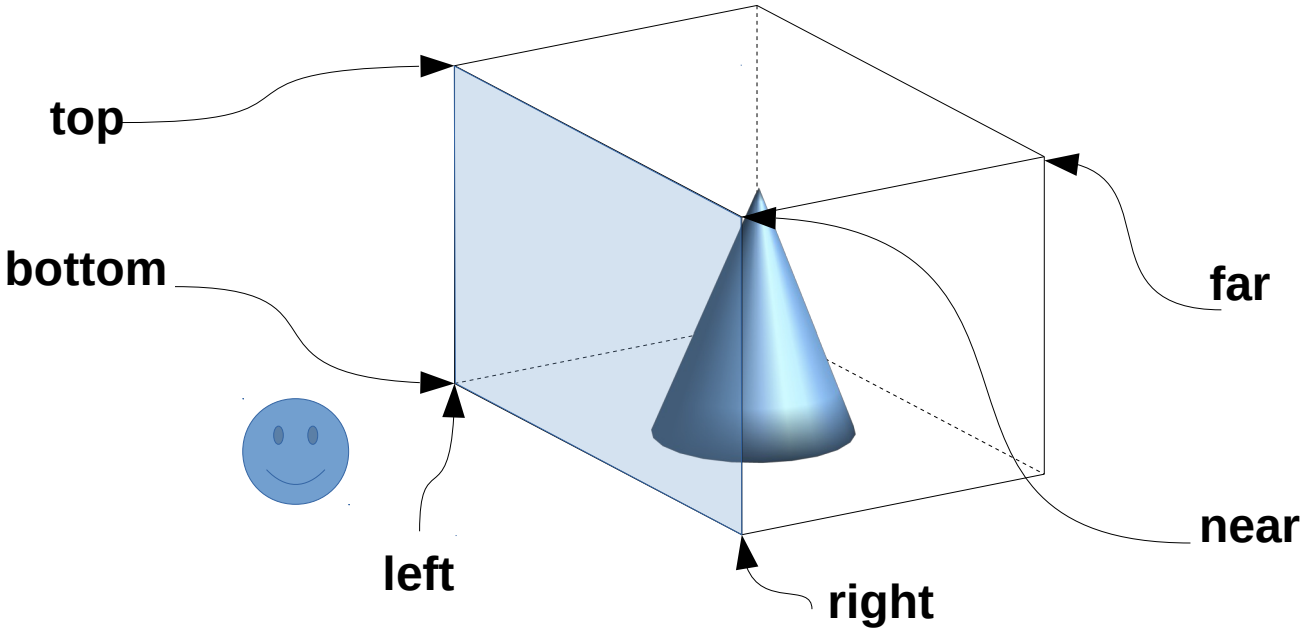
```
glLoadIdentity()
```

$$\begin{pmatrix} x_s \\ y_s \\ z_s \\ w_s \end{pmatrix} = M_p \cdot M_m * \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Проецирование

Ортогональное проецирование задается путем вызова функции **glOrtho**. Аргументы метода — параметры куба, внутри которого осуществляется проецирование. Объекты за пределами куба не отображаются.

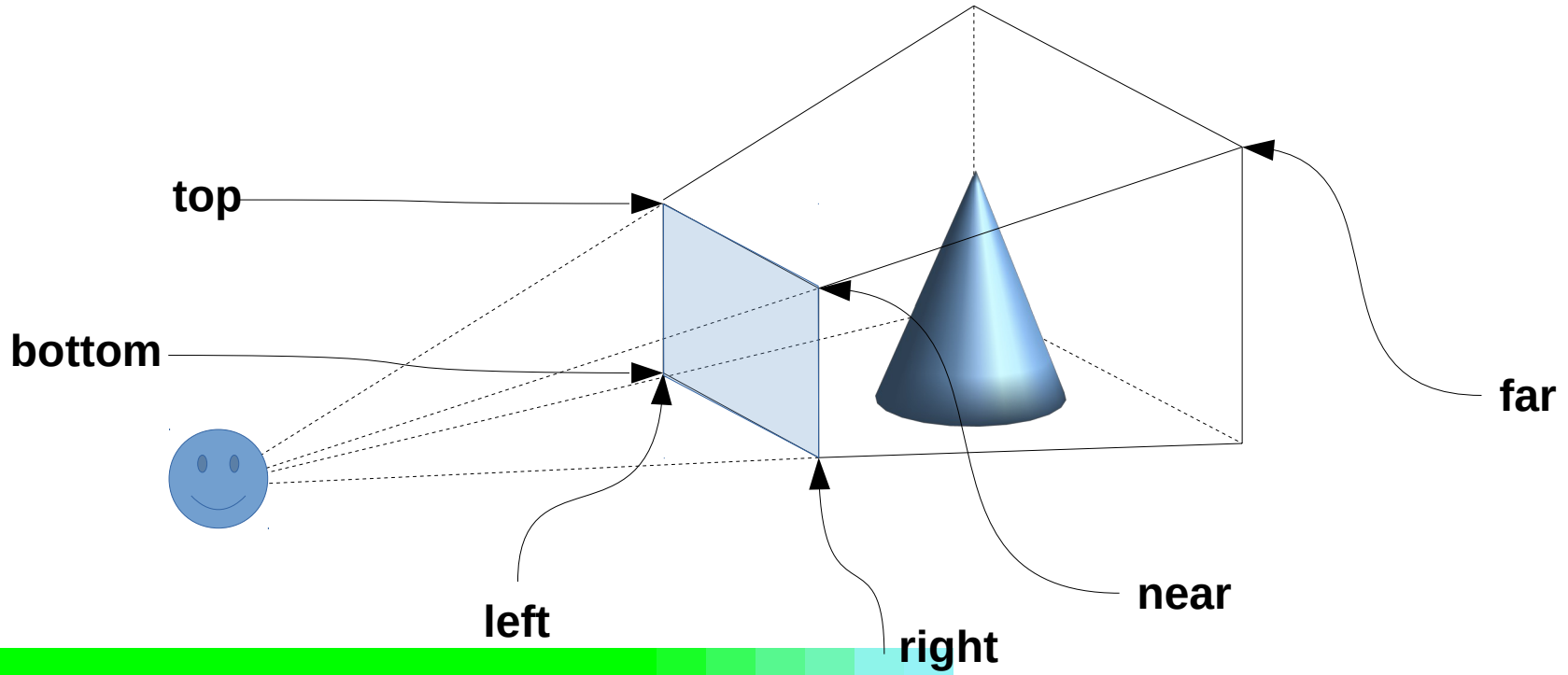
```
glOrtho(left, right, bottom, top, near, far)
```



Проецирование

Перспективное проецирование задается путем вызова функции **glFrustum**. Функция определяет усеченную пирамиду. Объекты внутри усеченной пирамиды проецируются на меньшее основание.

```
glFrustum(left, right, bottom, top, near, far)
```

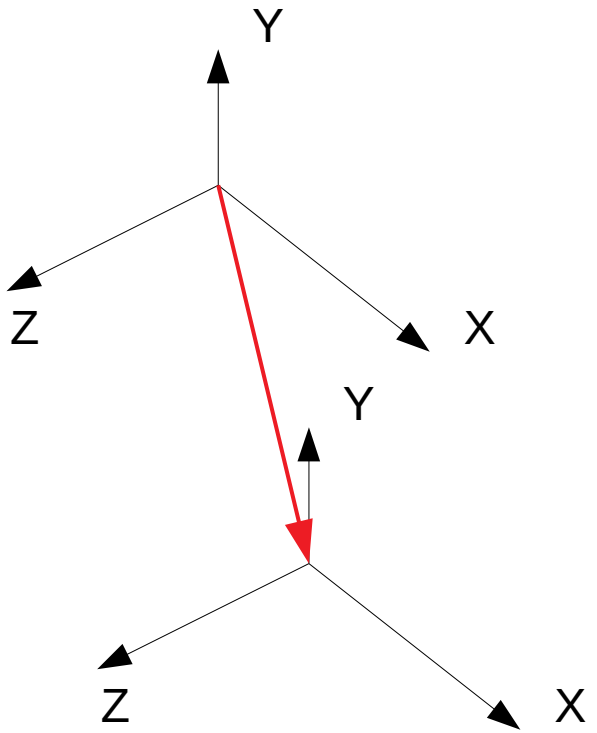


Трансформации

Параллельный перенос текущего базиса осуществляется функцией glTranslate. Она существует в двух формах: **glTranslated** и **glTranslatef** в зависимости от типа аргументов (**float** или **double**).

```
glTranslated(GLdouble x, GLdouble y, GLdouble z)
```

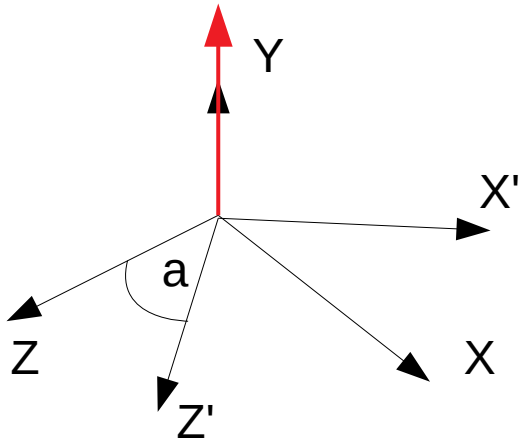
$$\begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Трансформации

Поворот текущего базиса осуществляется функцией glRotate. Она существует в двух формах: **glRotated** и **glRotatef**. Аргументы функции — это угол поворота в градусах и координаты вектора, вокруг которого происходит вращение.

```
glRotatef(GLfloat a, GLfloat x, GLfloat y, GLfloat z)
```

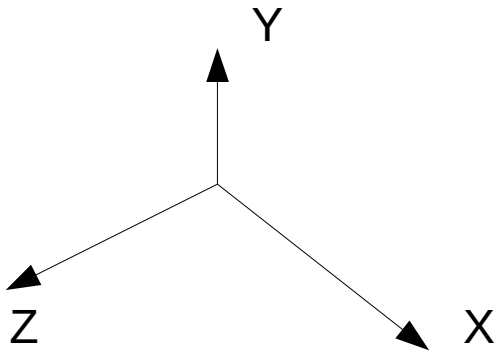


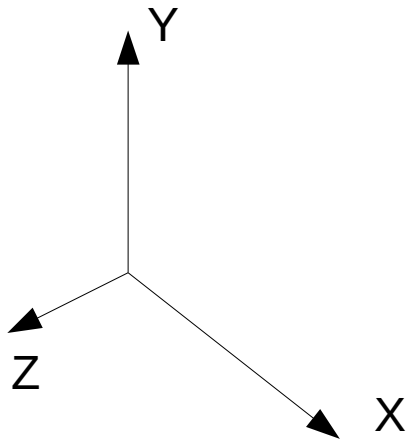
$$\begin{pmatrix} x^2+(1-x^2)\cos(a) & x\cdot y(1-\cos(a))+z\cdot\sin(a) & x\cdot z(1-\cos(a))-y\cdot\sin(a) & 0 \\ x\cdot y(1-\cos(a))-z\cdot\sin(a) & y^2+(1-y^2)\cos(a) & y\cdot z(1-\cos(a))+x\cdot\sin(a) & 0 \\ x\cdot z(1-\cos(a))+y\cdot\sin(a) & y\cdot z\cdot(1-\cos(a))-x\cdot\sin(a) & z^2+(1-z^2)\cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Трансформации

Масштабирование базиса осуществляется функцией glScale. Она существует в двух формах: **glScaled** и **glScalef**. Аргументы функции — это коэффициенты масштабирования по трем осям.

```
glScalef(GLfloat x, GLfloat y, GLfloat z)
```





$$\begin{pmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Матричные преобразования

При необходимости, можно сохранить текущую матрицу во внутренний стек, используя функцию **glPushMatrix()**. Последующий вызов **glPopMatrix()** вернет её на место.

```
glPushMatrix()              // Запоминание матрицы
glRotatef(30, 1, 0, 0)
// Рисование повернутого объекта
glPopMatrix()               // Возврат запомненной матрицы
```

Рисование

Отображаемые объекты задаются в виде отдельных вершин. Для определения вершины используется функция **glVertex**, имеющая множество вариантов записи.

```
glVertex2i(GLint x, GLint y)
glVertex3f(GLfloat x, GLfloat y, GLfloat z)
```

Помимо координат, вершина обладает такими свойствами как **цвет** и **нормаль**.

Нормаль задается функцией **glNormal**, которая также имеет множество вариантов записи. Непосредственная запись нормали происходит при вызове glVertex.

```
glNormal3f(0, 0, 1)
glVertex3f(3.3, 4, 2.12)
```

Рисование

Цвет задается при помощи функции **glColor** в виде трех компонентов (красный, зеленый и синий), либо четырех компонентов (+ альфа-канал).

При использовании вещественных чисел, минимальной интенсивности соответствует 0.0, а максимальной — 1.0.

При использовании целых чисел, используется весь диапазон целого числа.

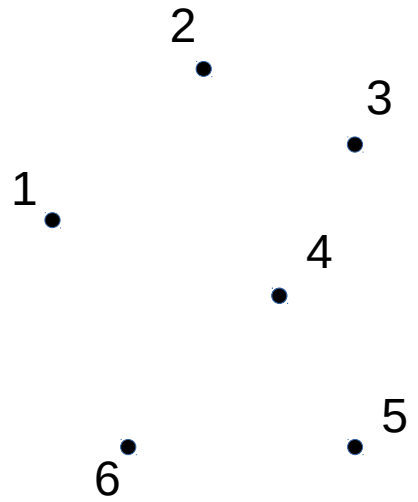
Непосредственная запись цвета происходит при вызове glVertex.

```
glColor4f(1.0, 0.0, 0.0, 0.5) // Красный полупрозрачный
glVertex3f(0, 0, 0)
glColor3ub(255, 255, 255)     // Белый
glVertex3f(3.3, 4, 2.12)
```

Рисование

Вызовы glVertex должны происходить между вызовами функций **glBegin** и **glEnd**. Аргумент glBegin определяет, как следует интерпретировать переданные вершины.

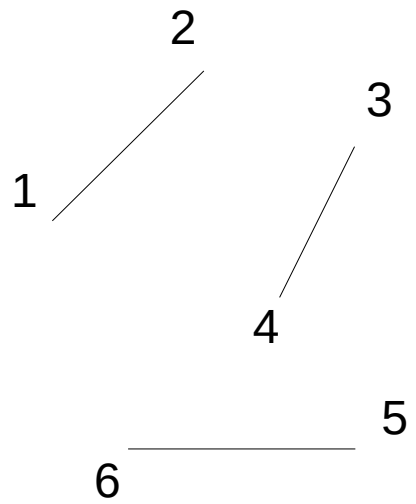
```
glBegin(GL_POINTS)
glVertex2f(0.0, 0.0)
glVertex2f(1.0, 1.0)
glvertex2f(2, 0.5)
glvertex2f(1.5, -0.5)
glvertex2f(2, -1.5)
glvertex2f(0.5, -1.5)
glEnd()
```



Рисование

Вызовы glVertex должны происходить между вызовами функций **glBegin** и **glEnd**. Аргумент glBegin определяет, как следует интерпретировать переданные вершины.

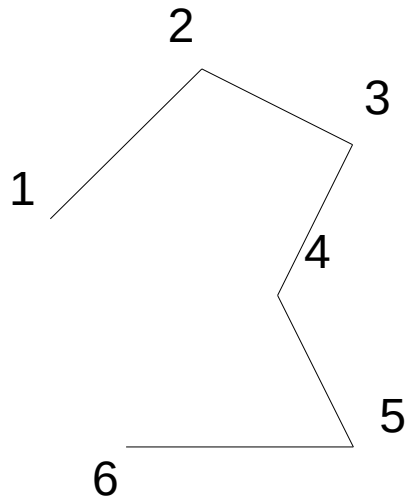
```
glBegin(GL_LINES)
glVertex2f(0.0, 0.0)
glVertex2f(1.0, 1.0)
glvertex2f(2, 0.5)
glvertex2f(1.5, -0.5)
glvertex2f(2, -1.5)
glvertex2f(0.5, -1.5)
glEnd()
```



Рисование

Вызовы glVertex должны происходить между вызовами функций **glBegin** и **glEnd**. Аргумент glBegin определяет, как следует интерпретировать переданные вершины.

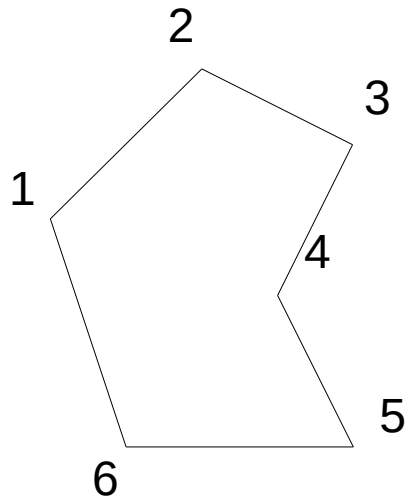
```
glBegin(GL_LINE_STRIP)
glVertex2f(0.0, 0.0)
glVertex2f(1.0, 1.0)
glvertex2f(2, 0.5)
glvertex2f(1.5, -0.5)
glvertex2f(2, -1.5)
glvertex2f(0.5, -1.5)
glEnd()
```



Рисование

Вызовы glVertex должны происходить между вызовами функций **glBegin** и **glEnd**. Аргумент glBegin определяет, как следует интерпретировать переданные вершины.

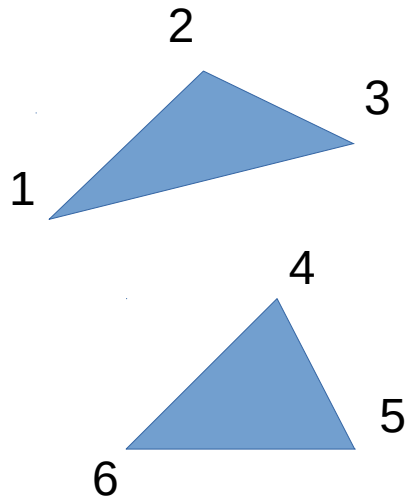
```
glBegin(GL_LINE_LOOP)
glVertex2f(0.0, 0.0)
glVertex2f(1.0, 1.0)
glvertex2f(2, 0.5)
glvertex2f(1.5, -0.5)
glvertex2f(2, -1.5)
glvertex2f(0.5, -1.5)
glEnd()
```



Рисование

Вызовы glVertex должны происходить между вызовами функций **glBegin** и **glEnd**. Аргумент glBegin определяет, как следует интерпретировать переданные вершины.

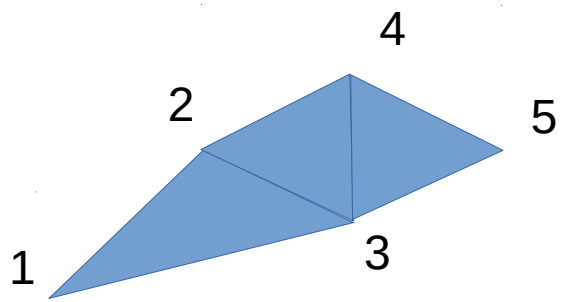
```
glBegin(GL_TRIANGLES)
glVertex2f(0.0, 0.0)
glVertex2f(1.0, 1.0)
glvertex2f(2, 0.5)
glvertex2f(1.5, -0.5)
glvertex2f(2, -1.5)
glvertex2f(0.5, -1.5)
glEnd()
```



Рисование

Вызовы glVertex должны происходить между вызовами функций **glBegin** и **glEnd**. Аргумент glBegin определяет, как следует интерпретировать переданные вершины.

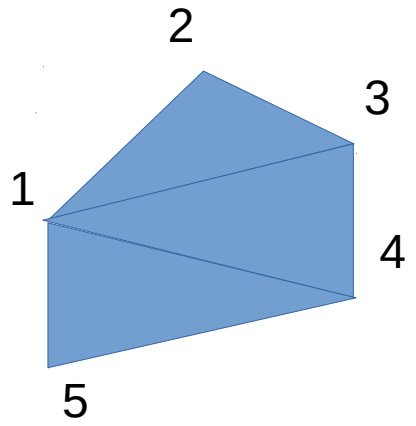
```
glBegin(GL_TRIANGLE_STRIP)
glVertex2f(0.0, 0.0)
glVertex2f(1.0, 1.0)
glvertex2f(2, 0.5)
glvertex2f(2, 1.5)
glvertex2f(3, 1)
glEnd()
```



Рисование

Вызовы glVertex должны происходить между вызовами функций **glBegin** и **glEnd**. Аргумент glBegin определяет, как следует интерпретировать переданные вершины.

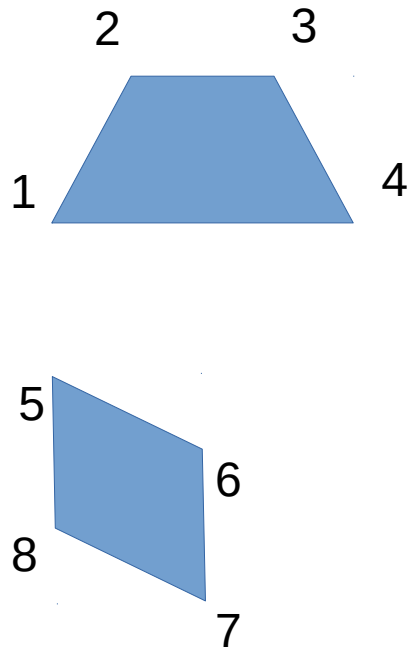
```
glBegin(GL_TRIANGLE_FAN)
glVertex2f(0.0, 0.0)
glVertex2f(1.0, 1.0)
glvertex2f(2, 0.5)
glvertex2f(2, -0.5)
glvertex2f(0, -1)
glEnd()
```



Рисование

Вызовы glVertex должны происходить между вызовами функций **glBegin** и **glEnd**. Аргумент glBegin определяет, как следует интерпретировать переданные вершины.

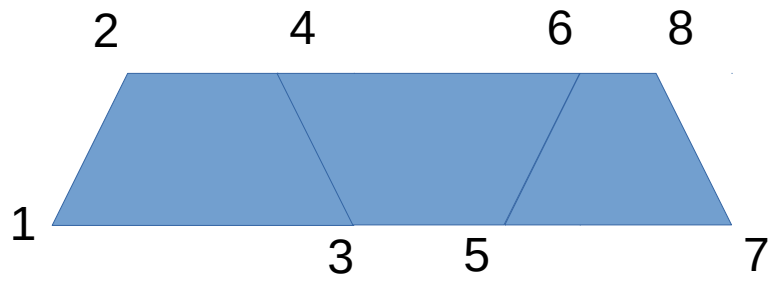
```
glBegin(GL_QUADS)
glVertex2f(0.0, 0.0)
glVertex2f(0.5, 1.0)
glvertex2f(1.5, 1)
glvertex2f(2, 0)
glvertex2f(0, -1)
glvertex2f(1, -1.5)
glvertex2f(1, -2.5)
glvertex2f(0, -2)
glEnd()
```



Рисование

Вызовы glVertex должны происходить между вызовами функций **glBegin** и **glEnd**. Аргумент glBegin определяет, как следует интерпретировать переданные вершины.

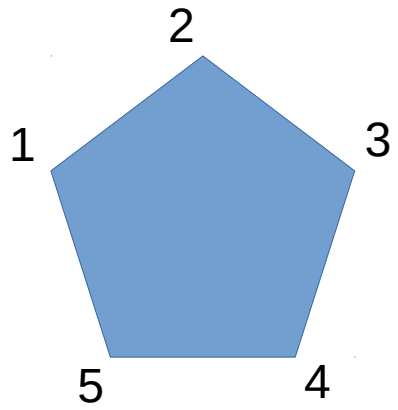
```
glBegin(GL_QUAD_STRIP)
glVertex2f(0.0, 0.0)
glVertex2f(0.5, 1.0)
glvertex2f(1.5, 1)
glvertex2f(2, 0)
glvertex2f(0, -1)
glvertex2f(1, -1.5)
glvertex2f(1, -2.5)
glvertex2f(0, -2)
glEnd()
```



Рисование

Вызовы glVertex должны происходить между вызовами функций **glBegin** и **glEnd**. Аргумент glBegin определяет, как следует интерпретировать переданные вершины.

```
glBegin(GL_POLYGON)
glVertex2f(0.0, 0.0)
glVertex2f(1.0, 0.8)
glvertex2f(0, 2)
glvertex2f(1.6, -1.2)
glvertex2f(0.4, -1.2)
glEnd()
```



Рисование

Для задания размера точки используется glPointSize. Влияет на GL_POINTS.

```
glPointSize(GLfloat w)
```

Для задания толщины линии используется функция glLineWidth. Влияет на GL_LINES, GL_LINE_STRIP и GL_LINE_LOOP.

```
glLineWidth(GLfloat w)
```

Функция glPolygonMode определяет способ рисования полигонов:

- GL_POINT — только вершины
- GL_LINE — только ребра
- GL_FILL — только заполнение

Освещение

Для включения освещения по Фонгу, необходимо использвать **glEnable** с константой **GL_LIGHTING**. Его также можно отключить через **glDisable** перед рисованием объектов, не требующих расчета освещения.

```
glEnable(GL_LIGHTING)
```

OpenGL поддерживает не менее 8 источников освещения, заданных константами GL_LIGHT0 .. GL_LIGHT7. Их также можно включать и выключать индивидуально.

```
glEnable(GL_LIGHT0)
```

Освещение

Свойства источника освещения задаются при помощи функции **glLight**. Используются варианты glLightfv для задания параметров-массивов или glLightf для одиночных параметров.

```
GLfloat pos[] = {1, 1, 1, 0};  // C++
pos = (1, 1, 1, 0);            # Python
glLightfv(GL_LIGHT0, GL_POSITION, pos);
```

Четвертый аргумент положения источника освещения определяет его тип. 0 — источник удален бесконечно далеко (солнечный свет), 1 — точечный источник.

Цвет освещения также задается через glLight для разных составляющих

```
GLfloat ambColor[] = {0.2, 0.2, 0.2, 0};  // C++
ambColor = (0.2, 0.2, 0.2, 0)             # Python
glLightfv(GL_LIGHT0, GL_AMBIENT, ambColor)
// GL_DIFFUSE и GL_SPECULAR для других составляющих
```

Освещение

Для корректной работы освещения необходимо задать свойства материала, используя функцию glMaterial. Более простой способ — использование цвета вершины в качестве параметров материала.

```
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT_AND_BACK,
                GL_AMBIENT_AND_DIFFUSE);
```

Также для правильной работы освещения обязательно нужно задавать нормаль для каждой вершины. Чтобы избежать искажений при трансформации и особенно при масштабировании, рекомендуется включить нормализацию, чтобы при всех расчетах длина нормали игнорировалась.

```
glEnable(GL_NORMALIZE);
```

Интеграция с Qt

Библиотека Qt содержит виджет QGLWidget, отображающий на своей поверхности результат работы OpenGL.

В C++ необходимо подключить модуль OpenGL, чтобы данный класс стал доступен.

```
QT += opengl
```

В Python достаточно импортировать модули:

```
from PyQt5.QtOpenGL import QGLWidget
from OpenGL.GL import *
```

QGLWidget не обладает какими либо функциями рисования. Для реализации рисования необходимо создать производный класс и переопределить методы из QGLWidget.

Наследование от QGLWidget

- **initializeGL()** — вызывается однократно при создании виджета. Может использоваться для загрузки текстур.
- **resizeGL(int, int)** — вызывается при изменении размеров виджета. Используется для задания размера отображаемой области.
- **paintGL()** — используется непосредственно для рисования сцены OpenGL.

Функции OpenGL недоступны за пределами данных методов.

При необходимости обратиться к контексту OpenGL в другом месте следует вызвать метод **makeCurrent()**.

Для принудительной перерисовки окна следует вызвать слот **updateGL**.

Создание приложения OpenGL

1. Создать основу приложения.
2. Создать класс, производный от QGLWidget и разместить его в окне.
3. В процедуре рисования, сбросить матрицу проецирования (glLoadIdentity), затем задать проецирование (glOrtho, glFrustum) и сделать преобразования, определяющие ракурс наблюдения объекта.
4. Переключиться на матрицу моделирования. Сбросить матрицу и применить преобразования, описывающие поворот и перемещение объекта.
5. * Задать свойства освещения.
6. Нарисовать объект путем вызова glVertex между glBegin и glEnd.

Дополнительные материалы

1. Текстурирование: glGenTextures, glBindTexture, glTexImage2D.
2. Vertex Buffer Object
3. GLSL
4. Adrian Courrèges
   - GTA V Graphics Study

     http://www.adriancourreges.com/blog/2015/11/02/gta-v-graphics-study/
   - DOOM 2016 Graphics Study

     http://www.adriancourreges.com/blog/2016/09/09/doom-2016-graphics-study/