LOBACHEVSKY STATE UNIVERSITY
of NIZHNI NOVGOROD
National Research University

Computing Mathematics and Cybernetics faculty
Software department

Computer Graphics. Introduction Course

OpenGL

Белокаменская А.А., Васильев Е.П.

Что такое OpenGL (Open Graphics Library)

- OpenGL is an API (Application Programming Interface) to graphics hardware.
- The API consists of a set of several hundred procedures and functions that allow a programmer to specify the shader programs, objects, and operations involved in producing high-quality graphical images, specifically color images of three-dimensional objects.

(Спецификация)

Что такое OpenGL (Open Graphics Library)

- OpenGL – это спецификация, определяющая независимый от языка программирования платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трёхмерную компьютерную графику.

  (Википедия)

Что такое OpenGL (Open Graphics Library)

- Конвейер, который включает в себя несколько программируемых этапов, и несколько фиксированных.

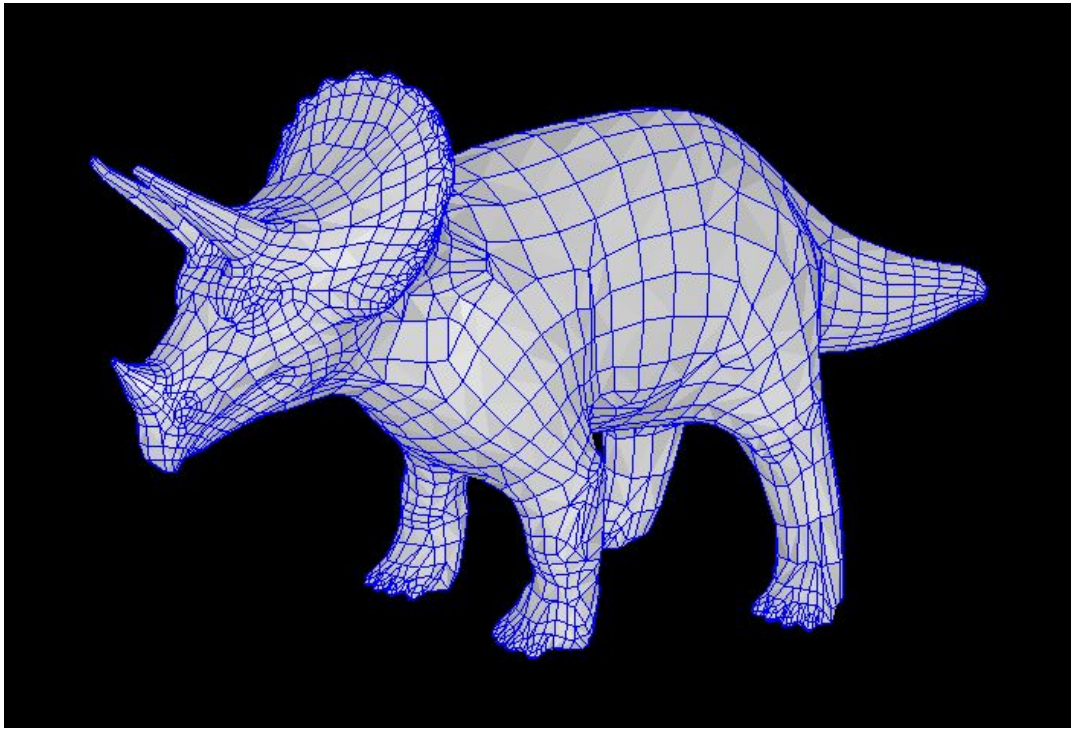
(Консорциум)

OpenGL API

- As a programmer, you need to do the following things:
  - Specify the location/parameters of camera.
  - Specify the geometry (and appearance).
  - Specify the lights (optional).

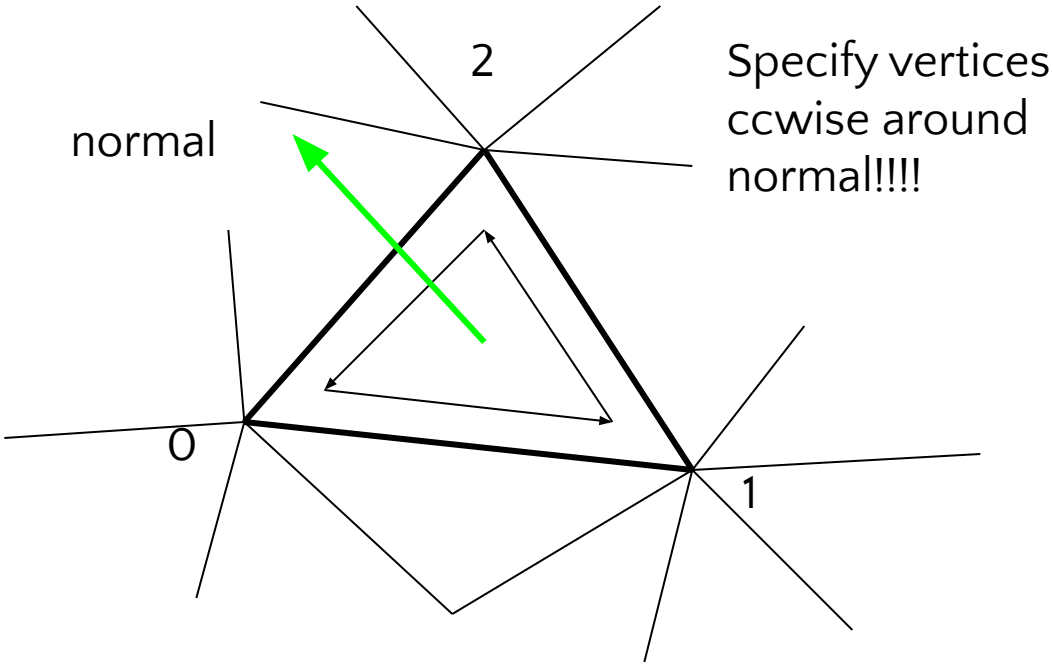
OpenGL: Geometry

- Specify geometry using primitives: triangles, quadrilaterals, lines, points, etc…



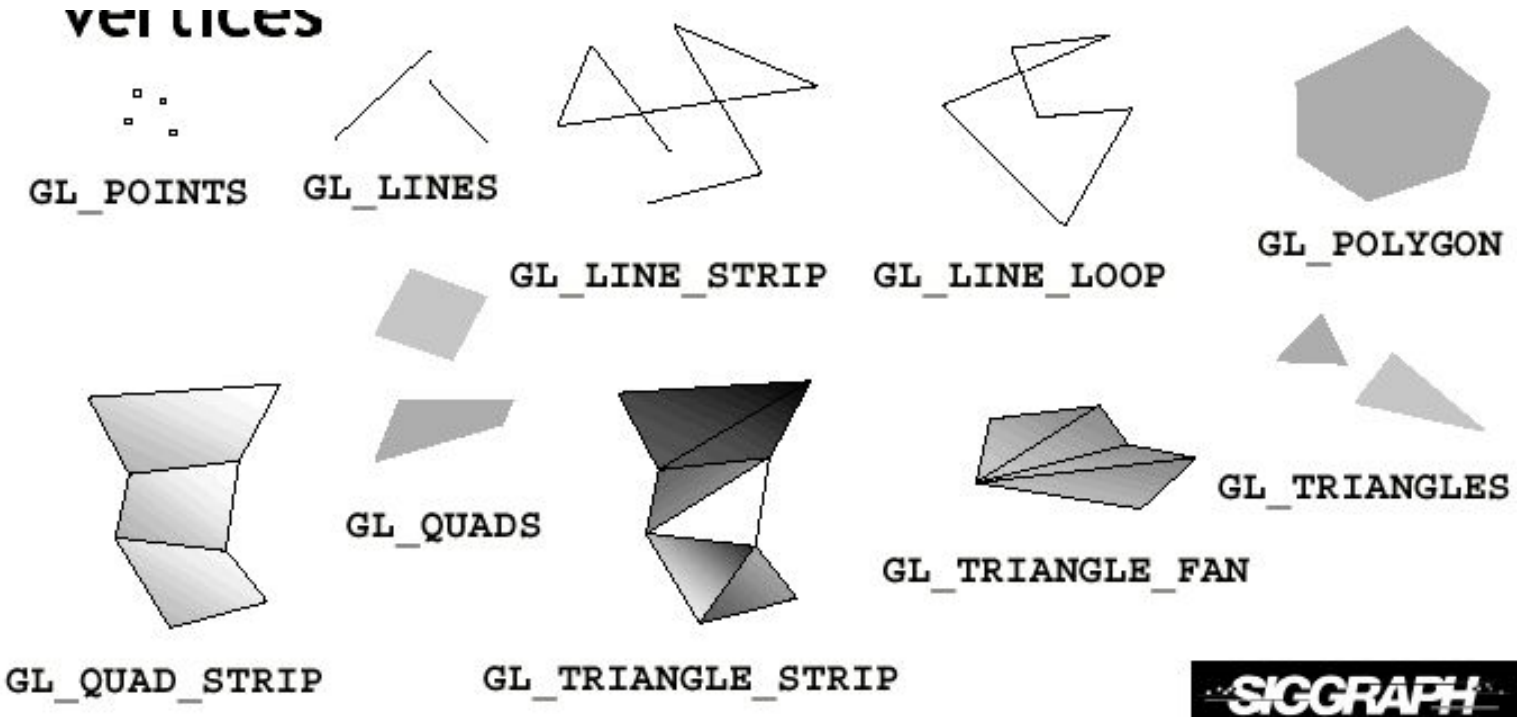
OpenGL: Triangle Mesh

- Represent the surface of an object by a collection of **oriented** triangles:

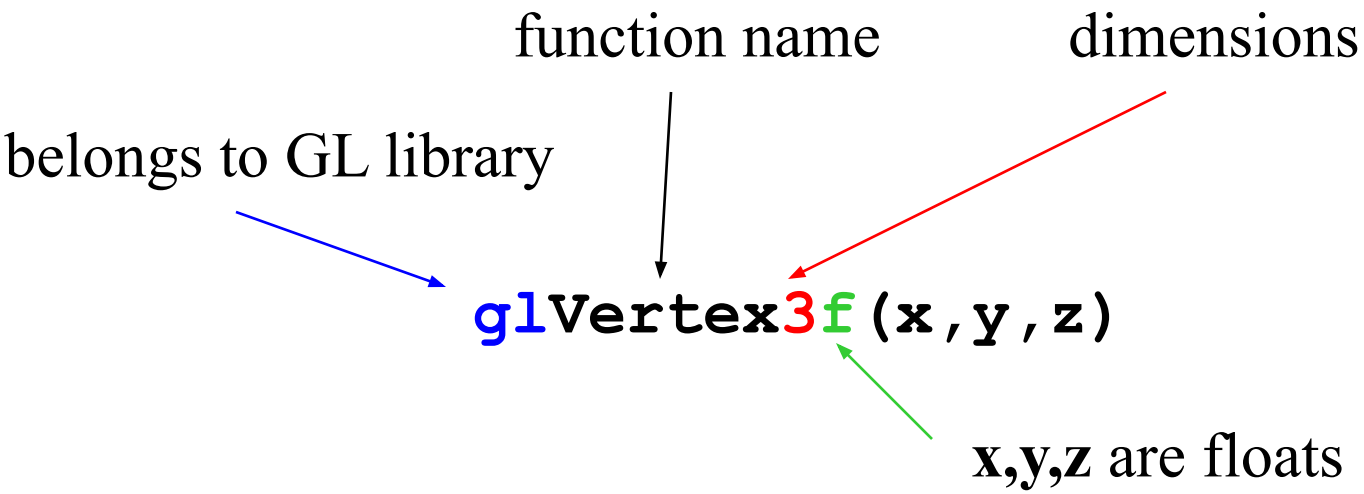


OpenGL Geometric Primitives

- All geometric primitives are specified by vertices



OpenGL function format




```
glVertex3fv(p)
```

p is a pointer to an array

OpenTK function format

- GL.Vertex2(–1, –1, 1); +17 перегруженных
- GL.Vertex3(–1, –1, 1); +17 перегруженных
- GL.Vertex4(–1, –1, 1); +17 перегруженных

OpenGL: glBegin()…glEnd()

- Точки геометрии прописываются между «скобками» glBegin(…), glEnd()

```
glBegin(GL_TRIANGLES);
for (int i=0; i<ntris; i++)
{
    glColor3f(tri[i].r0,tri[i].g0,tri[i].b0);          // Color of vertex
    glNormal3f(tri[i].nx0,tri[i].ny0,tri[i].nz0);    // Normal of vertex
    glVertex3f(tri[i].x0,tri[i].y0,tri[i].z0);         // Position of vertex
    …
}
glEnd();   // Sends all the vertices/normals to the OpenGL library
```

OpenTK:

- GL.Begin(PrimitiveType.Quads);

OpenGL: glBegin()…glEnd()

- OpenGL supports many primitives:

  glBegin(GL_LINES);

  glBegin(GL_QUADS);

  glBegin(GL_POLYGON);

- OpenTK содержит перечисление PrimitiveTypes

glEnable() и glDisable()

- Включают и выключают различные возможности OpenGL

void glEnable(GLenum cap);

void glDisable(GLenum cap);

cap:

GL_BLEND, GL_CULL_FACE, GL_DEPTH_TEST, GL_LINE_SMOOTH, GL_POLYGON_SMOOTH

В OpenTK для этого есть перечисление **`EnableCap`**

void glHint(GLenum target, GLenum mode);

- Некоторые аспекты поведения OpenGL могут иметь дополнительные настройки с помощью glHint()
- *Target*
  - GL_FOG_HINT, GL_GENERATE_MIPMAP_HINT, GL_LINE_SMOOTH_HINT, GL_PERSPECTIVE_CORRECTION_HINT, GL_POINT_SMOOTH_HINT, GL_POLYGON_SMOOTH_HINT, GL_TEXTURE_COMPRESSION_HINT, GL_FRAGMENT_SHADER_DERIVATIVE_HINT
- *Mode*
  - GL_FASTEST, GL_NICEST, GL_DONT_CARE

SwapBuffers();

- К контексту OpenGL принадлежит два цветных буфера, один для отрисовки, другой для отображения его на экране. SwapBuffers() меняет их местами
- На самом деле буферов, в том числе и цветных больше, ещё есть буфер глубины, маски.
- Перед отрисовкой каждого кадра эти буферы надо очистить с помощью glClear();

Камера

- **Видовая трансформация**
- gluLookAt (eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)
- По умолчанию
  - Позиция камеры (0, 0, 0)
  - Направление вдоль отрицательного направления оси z
  - Вектор верхнего направления (0, 1, 0)

Объекты

- **Модельная трансформация**
  - Translation: glTranslate(x,y,z)
  - Scale: glScale(sx,sy,sz)
  - Rotation: glRotate(theta, x,y,z)

Проекционная трансформация

- Перспективная проекция
- Ортографическая проекция

Трансформация порта просмотра

- Задает форму и размеры доступной области на экране, куда будет перенесено изображение
- glViewport() задает начальную точку доступного экранного пространства внутри окна, а также ширину и высоту доступной области на экране.

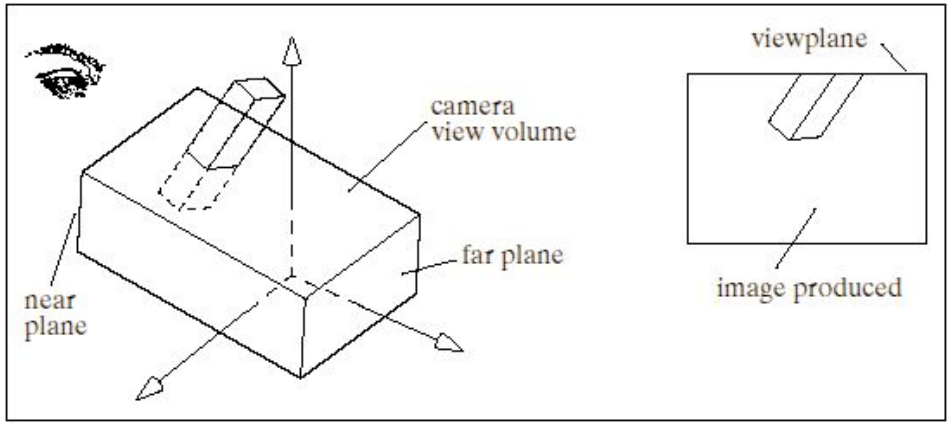
Модельно-видовая матрица

- Видовые и модельные преобразования в OpenGL объединены в одной матрице.
- для достижения определенной композиции вы можете либо перемещать камеру, либо перемещать все объекты сцены в противоположном направлении.
- Модельное преобразование, поворачивающее объекты сцены против часовой стрелки аналогично видовому преобразованию, которое поворачивает камеру по часовой стрелке.
- Команды видового преобразования должны вызываться перед всеми командами модельных преобразований, чтобы модельные преобразования были применены к объектам первыми.

Своя модель камеры

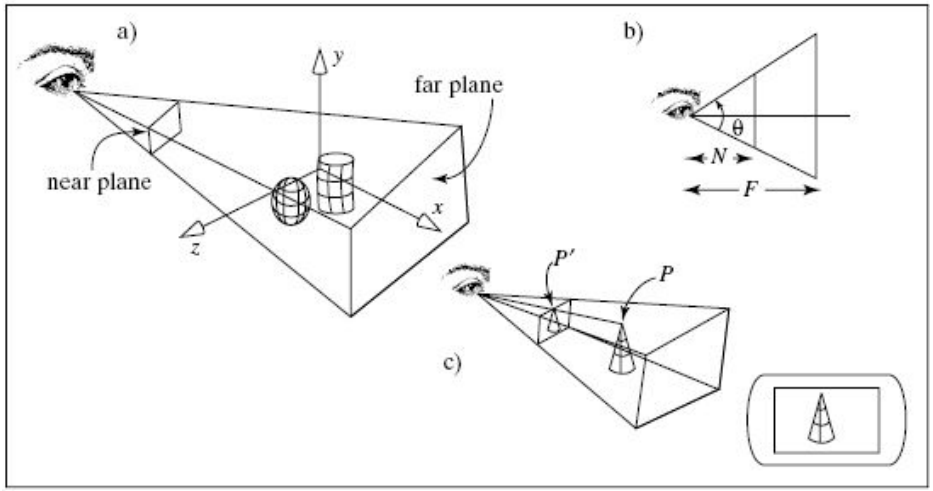
- В некоторых приложениях может понадобиться функция, чтобы можно было задавать видовую трансформацию каким-либо специфическим путем.
- Например, вам может понадобиться задавать преобразование в терминах полярных координат для камеры, вращающейся вокруг объекта или в терминах углов наклона самолета в полете.

Projection Transformation



Orthographic projection

Perspective projection



Transformations in OpenGl

- Modeling transformation
  - Refer to the transformation of models (i.e., the scenes, or objects)
- Viewing transformation
  - Refer to the transformation on the camera
- Projection transformation
  - Refer to the transformation from scene to image

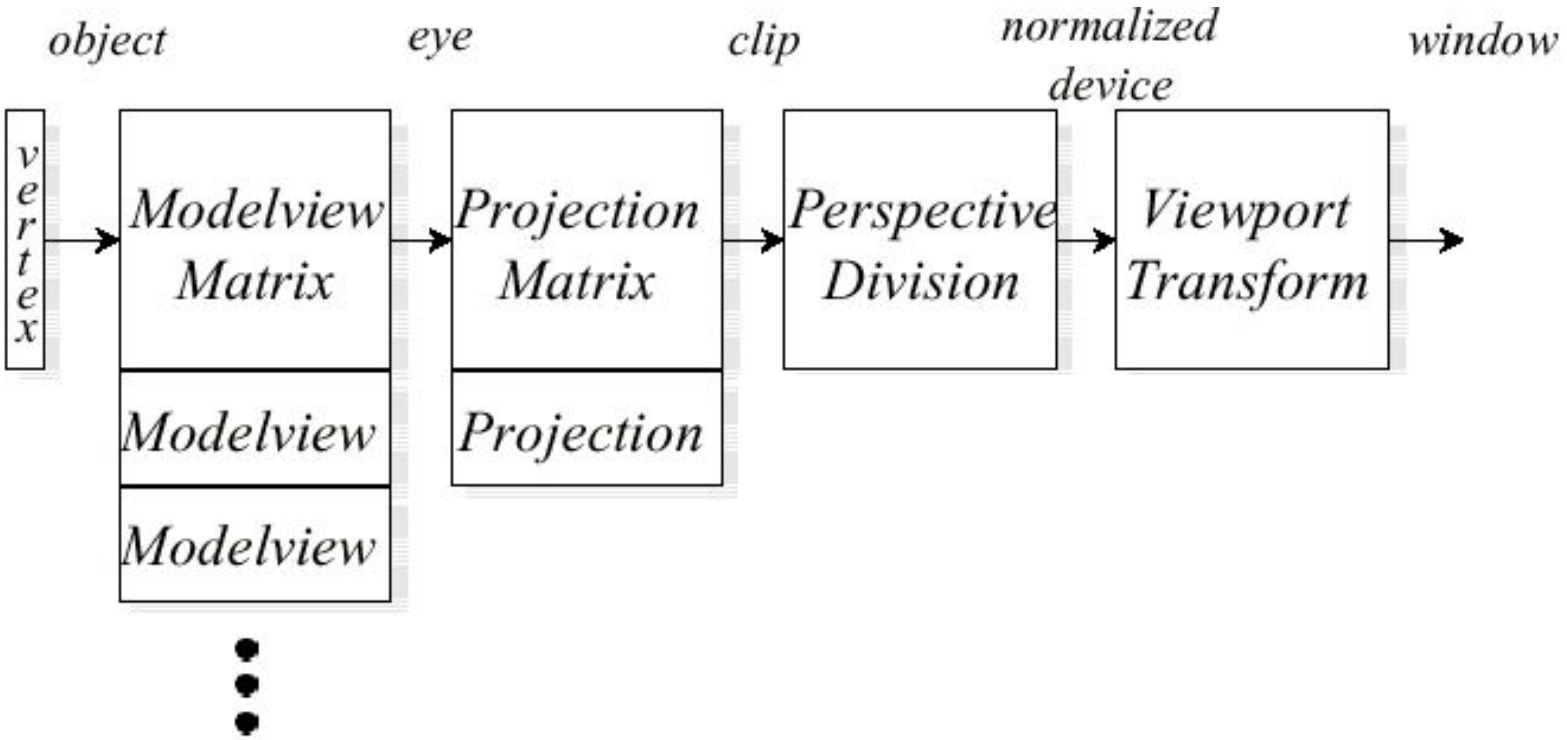
Model/View Transformations

- Model-view transformations are usually visualized as a single entity
  - Before applying modeling or viewing transformations, need to set glMatrixMode(GL_MODELVIEW)
  - Modeling transforms the object
    - Translation: glTranslate(x,y,z)
    - Scale: glScale(sx,sy,sz)
    - Rotation: glRotate(theta, x,y,z)
  - Viewing transfers the object into camera coordinates
    - gluLookAt (eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ)

Projection Transformation

- Transformation of the 3D scene into the 2D rendered image plane
  - Before applying projection transformations, need to set glMatrixMode(GL_PROJECTION)
  - Orthographic projection
    - glOrtho(left, right, bottom, top, near, far)
  - Perspective projection
    - glFrustum (left, right, bottom, top, near, far)

Transformation Pipeline



Matrix Operations

- Specify Current Matrix Stack
  - **glMatrixMode(GL_MODELVIEW or GL_PROJECTION)**
- Other Matrix or Stack Operation
  - **glLoadIdentity() glPushMatrix() glPopMatrix()**
- Viewport
  - usually same as window size
  - viewport aspect ratio should be same as projection transformation or resulting image may be distorted
  - **glViewport(x, y, width, height)**

Projection Transformation

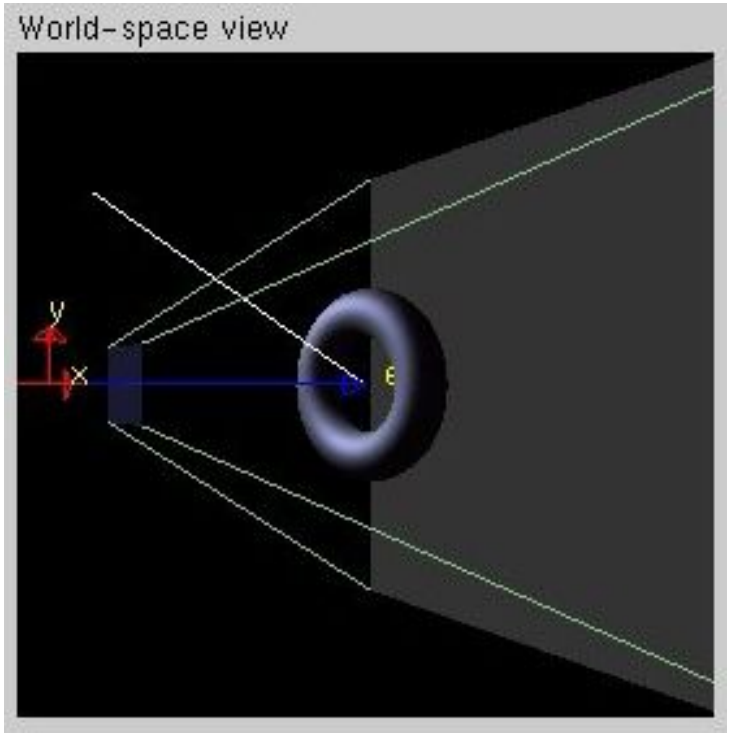
- Perspective projection
  - **gluPerspective(fovy, aspect, zNear, zFar)**
  - **glFrustum(left, right, bottom, top, zNear, zFar)** (very rarely used)
- Orthographic parallel projection
  - **glOrtho(left, right, bottom, top, zNear, zFar)**
  - **gluOrtho2D(left, right, bottom, top)**
  - calls **glOrtho** with z values near zero
- *Warning:* for `gluPerspective()` or `glFrustum()`, don't use zero for `zNear`!

Projection Transformation OpenTK

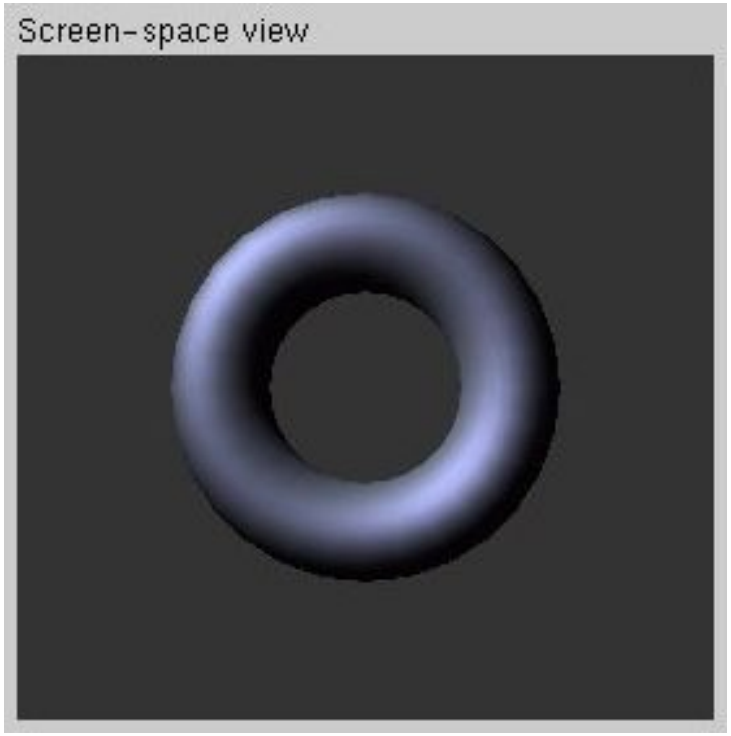
```
GL.MatrixMode(MatrixMode.Projection);
Matrix4 projection =
  Matrix4.CreatePerspectiveFieldOfView(
   MathHelper.PiOver4,
   width / (float)height,
   0.1f, 100.0f);
GL.LoadMatrix(ref projection);

Matrix4 ortho =
  Matrix4.CreateOrthographic(width,height,
  znear, zfar);
```

OpenGL: MODELVIEW

World coord-sys:

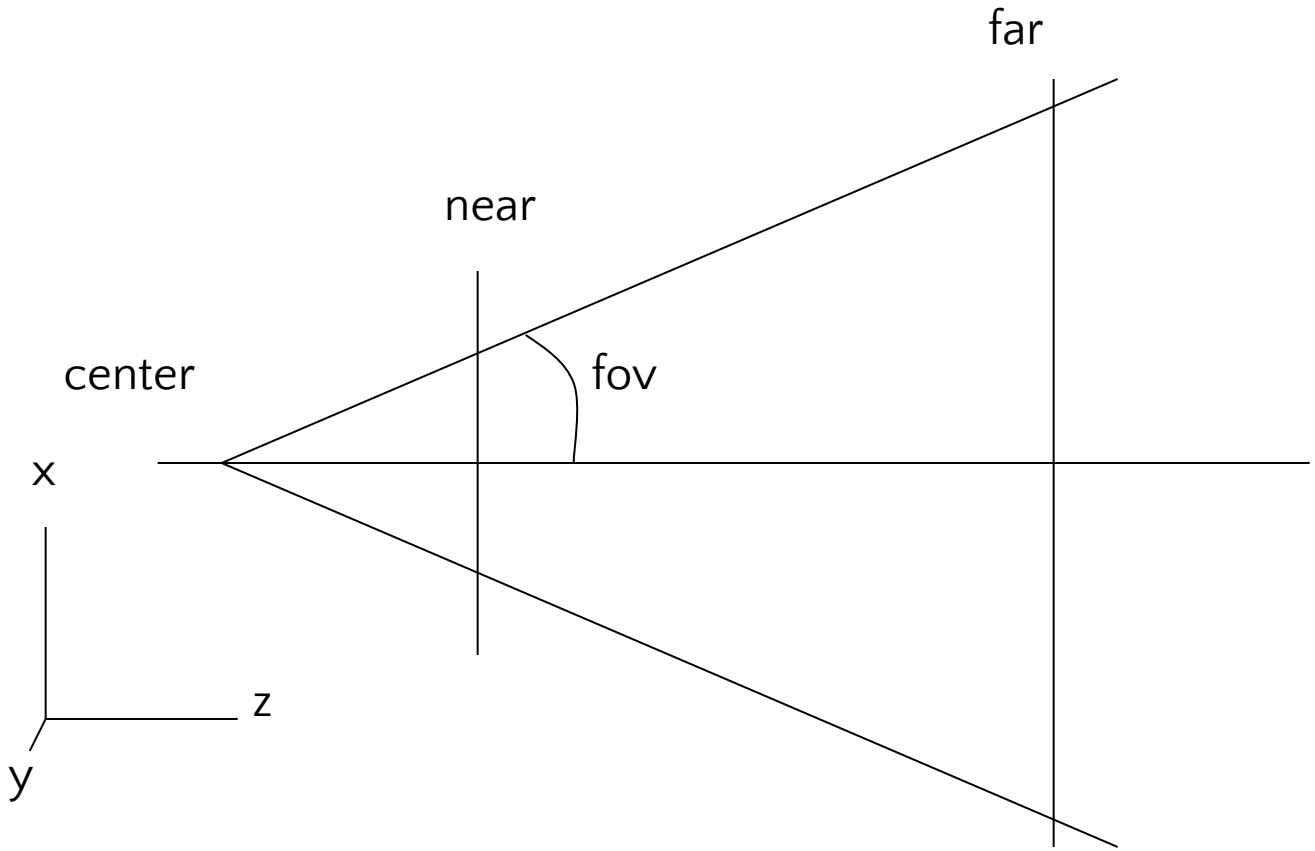


Camera coord-sys:



OpenGL: PROJECTION

- Intrinsic (optical) properties of camera:

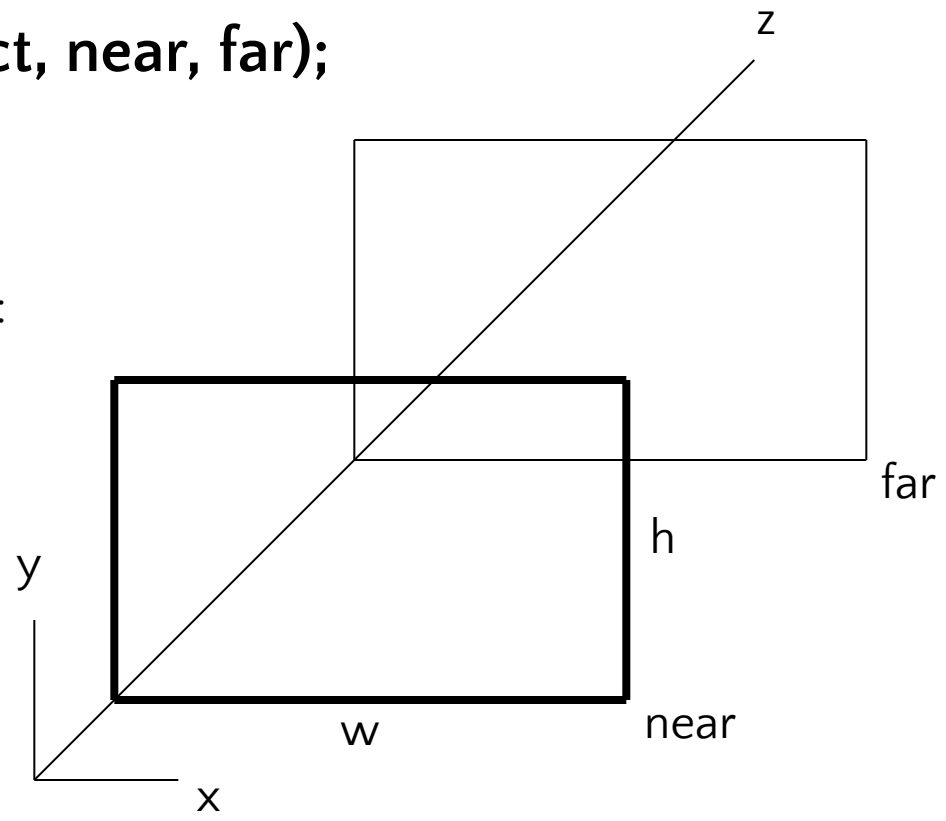


OpenGL: PROJECTION

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(fovy, aspect, near, far);
```

Camera viewport:

aspect = w/h



OpenGL: Setting Camera

- Assume window is **width×height**:

```
void SetCamera()
{
  glViewport(0, 0, width, height);
  /* Set camera position */
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(m_vEye[0], m_vEye[1], m_vEye[2],
   m_vRef[0], m_vRef[1], m_vRef[2],
       m_vUp[0],  m_vUp[1],  m_vUp[2]);
  /* Set projection frustrum */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(m_fYFOV, width / height, m_fNear,
  m_fFar);
}
```

Освещение

- Метод трассировки лучей, и метод излучательности требуют большого объема вычислений, поэтому основное внимание уделяется более простым локальным моделям заполнения, основанным на модели отражения Фонга (Phong)

Типы взаимодействия света и материала поверхности

- Зеркальное отражение. Поверхности выглядят блестящими, т.к. большая часть световой энергии отражается или рассеивается в узком диапазоне углов, близких к углу отражения.
- Диффузное отражение. При диффузном отражении падающий свет рассеивается в разных направлениях.
- Преломление. Луч света, падающий на поверхность, преломляется и проникает в среду объекта под другим углом. Как правило, при этом отражается часть падающего света.

Источник

- любой источник рассматривается, как состоящий из трех независимых источников первичных цветов и соответственно его описывает трехкомпонентная функция излучения:

$$I = \begin{bmatrix} I_r \\ I_g \\ I_b \end{bmatrix}$$

Освещение

- OpenGL рассчитывает свет и освещение так, как будто свет может быть разделен на красный, зеленый и синий компоненты.
- источник света характеризуется количеством красного, зеленого и синего света, которое он излучает
- материал поверхности характеризуется долями красного, зеленого и синего компонентов, которые он отражает в различных направлениях.

4 компоненты освещения (задается материалом)

- *фоновое* (*ambient)* свет, который настолько распределен в среде, что его направление определить невозможно. Когда фоновый свет падает на поверхность, он одинаково распределяется во всех направлениях
- *диффузное (diffuse)* свет, идущий из одного направления, таким образом, он выглядит ярче, если падает на поверхность под прямым углом, и выглядит тусклым, если касается ее всего лишь вскользь. Когда он падает на поверхность, он распределяется одинаково во всех направлениях
- *зеркальное (specular)* исходит из определенного направления и отражается от поверхности в определенном направлении.
- *исходящее* (эмиссионное – *emissive)* свет, исходящий от самого объекта. Добавляет объекту интенсивности, но на него не влияют никакие источники света, и он не производит дополнительного света для сцены в целом.

Компоненты освещения для источника

- ambient
- diffuse
- specular

Материал

- материалы имеют разные фоновый, диффузный и зеркальный цвета, которые задают реакцию материала на фоновый, диффузный и зеркальный компоненты света.
- Фоновый цвет материала комбинируется с фоновым компонентом всех источников света, диффузный цвет с диффузным компонентом, а зеркальный с зеркальным.
- Фоновый и диффузный цвета задают видимый цвет материала, они обычно близки, если не эквивалентны.
- Зеркальный цвет обычно белый или серый.

Подключение источников света OpenTK

- Доступны не менее 8 источников света

```
GL.Enable(EnableCap.Lighting);
GL.Enable(EnableCap.Light0);
```

Источники направленного света

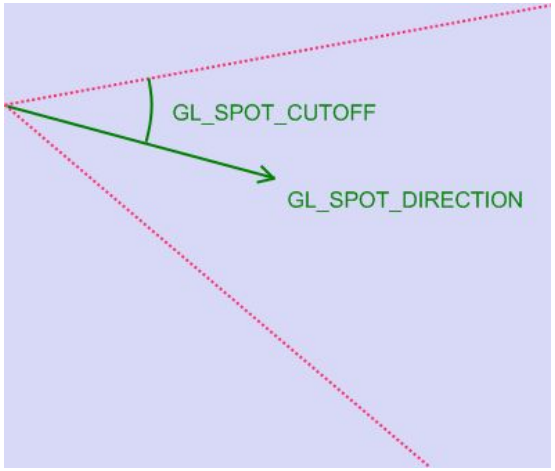
- Источник света такого типа находится в бесконечности и свет от него распространяется в заданном направлении.
- GL_POSITION по умолчанию (0.0, 0.0, 1.0, 0.0) – направление источника направленного света
- Первые три компоненты *(x, y, z)* задают вектор направления, а компонента *w* всегда равна нулю (иначе источник превратится в точечный).

Точечные источники света

- Точечный источник света расположен в некоторой точке пространства и излучает во всех направлениях.
- закон убывания интенсивности излучения с расстоянием в виде обратно-квадратичной функции от расстояния: fatt(d) = 1/(kconst + klinear*d+kquadratic*d*d)
- Коэффициенты задаются параметрами
  - GL_CONSTANT_ATTENUATION
  - GL_LINEAR_ATTENUATION
  - GL_QUADRATIC_ATTENUATION

Прожекторы

- прожектор позволяет ограничить распространение света конусом
- можно задать коэффициент убывания интенсивности, в зависимости от угла между осью конуса и лучом распространения света



Свойства источника света OpenTK

```
GL.Light(LightName.Light0, LightParameter.Ambient, light_ambient);
GL.Light(LightName.Light0, LightParameter.Diffuse, light_diffuse);
GL.Light(LightName.Light0, LightParameter.Specular, light_specular);

// Постоянный фактор ослабления
GL.Light(LightName.Light0, LightParameter.ConstantAttenuation, 1.8f);
// Угловая ширина светового луча
GL.Light(LightName.Light0, LightParameter.SpotCutoff, 45.0f);
// Направление света прожектора
GL.Light(LightName.Light0, LightParameter.SpotDirection, spotdirection);
// Концентрация светового луча
GL.Light(LightName.Light0, LightParameter.SpotExponent, 0.0f);

GL.LightModel(LightModelParameter.LightModelLocalViewer, 1.0f);
GL.LightModel(LightModelParameter.LightModelTwoSide, 1.0f);
```

Свойства источника света OpenTK

- GL_LIGHT_MODEL_LOCAL_VIEWER в графической системе устанавливает режим "близкого" наблюдателя. Если наблюдатель расположен далеко от рассматриваемой сцены, то можно считать, что вектор, задающий направление на наблюдателя, для всех объектов сцены один и тот же. Это сокращает объем вычислений.
- Если наблюдатель с определенной позиции может "заглянуть" внутрь объекта и увидеть внутренние грани, тогда для корректного закрашивания нужно установить в true параметр GL_LIGHT_MODEL_TWO_SIDED

Свойства источника света

- геометрические параметры источников света преобразуются матрицей вида, поэтому можно задавать их положение, используя привычные средства преобразования.

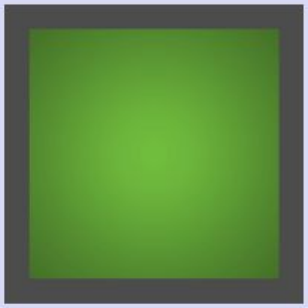
Глобальное фоновое освещение

- В сцену можно включить и глобальное фоновое освещение, которое не связано ни с каким отдельным источником. Если, например требуется слабо подсветить все объекты сцены белым цветом, в программу следует включить такой фрагмент кода:

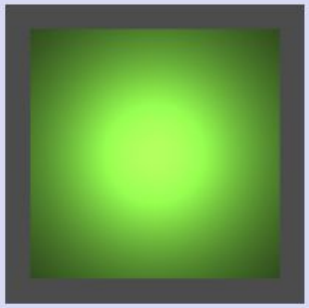
```
GLfloat global_ambient[]={0.1, 0.1, 0.1, 1.0};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient);
```

Пример

Направленный источник света

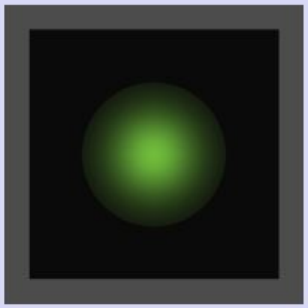


Точечный источник света, убывание интенсивности с расстоянием выключено



Точечный источник света, убывание интенсивности с расстоянием включено

Прожектор



Прожектор, включен расчет убывания интенсивности для прожектора

Несколько источников света

Установка параметров материала

- void glMaterial{if}(GLenum face, GLenum pname, TYPE param);
- Аргумент *face* указывает для каких граней объекта задается свойство материала: *GL_FRONT, GL_BACK GL_FRONT_AND_BACK*

Установка параметров материала

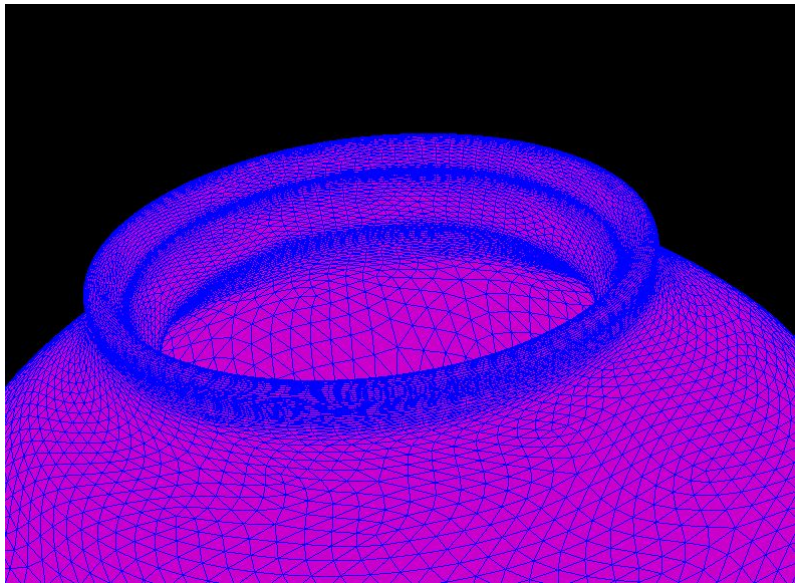
- GL_AMBIENT (0.2, 0.2, 0.2, 1.0) цвет фонового отражения материала
- GL_DIFFUSE (0.8, 0.8, 0.8, 1.0) цвет рассеянного отражения материала
- GL_SPECULAR (0.0, 0.0, 0.0, 1.0) цвет зеркального отражения материала
- GL_EMISSION (0.0, 0.0, 0.0, 1.0) цвет собственного излучения материала
- GL_SHININESS 0.0 степень в формуле зеркального отражения (коэффициент блеска). Допускаются значения в интервале [0; 128].
- GL_AMBIENT_AND_DIFFUSE цвет фонового и рассеянного отражения материала

Установка параметров материала

- По умолчанию, при включении расчета освещения, текущий цвет, задаваемый командой glColor4f, игнорируется.
- Можно включить управление свойством материала с помощью текущего цвета, т.е. изменять одну из характеристик отражения материала командой glColor4f()
- glEnable(GL_COLOR_MATERIAL);
- glColorMaterial(GLenum face, GLenum mode); задает, какое конкретно свойство материала будет передаваться текущим цветом

OpenGL: Flat Shading

- В режиме плоского закрашивания OpenGL использует вектор нормали, ассоциированный с первой вершиной каждого очередного закрашиваемого многоугольника
- glShadeModel(GL_FLAT);



OpenGL: Flat Shading

- OpenGL будет интерполировать цвет вдоль отображаемого примитива.
- glShadeModel(GL_SMOOTH);

Текстуры

- Текстуры – это прямоугольные массивы данных, например, цветовых, световых или цветовых и альфа. Индивидуальные элементы (значения) текстуры часто называются *текселями (texels).*
- *Что делает текстурирование сложным, так это то, что* прямоугольная текстура может быть наложена на непрямоугольный объект, и это должно быть сделано каким-либо разумным способом.

Наложение текстуры

- Чтобы использовать наложение текстуры, вы должны выполнить следующие шаги:
  - Создать текстурный объект и задать текстуру для него
  - Задать, как текстура должна воздействовать на каждый пиксель
  - Активизировать механизм текстурирования
  - Нарисовать сцену, передавая на конвейер визуализации и геометрические координаты и координаты текстуры
- Текстурирование работает только в RGBA режиме. Результат попытки применения текстурирования в индексном режиме не определен.

Наложение текстуры

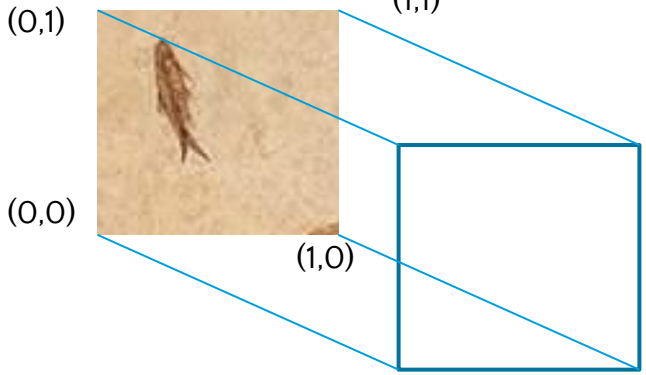
- glEnable() (glDisable())
  - GL_TEXTURE_1D
  - GL_TEXTURE_2D
  - GL_TEXTURE_3D

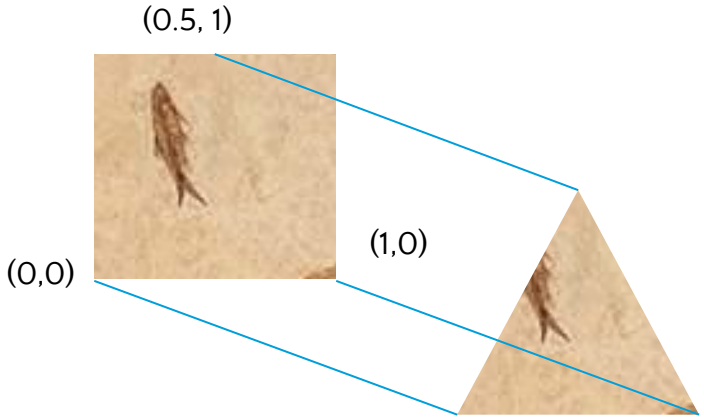
Загрузка текстуры в память

- void glTexImage(1/2/3)D(GLenum target, GLint level, GLint internalFormat, GLsizei width, (GLsizei height), GLint border, GLenum format, GLenum type, const GLvoid * data)
- **target**: GL_TEXTURE_(1/2/3)D
- **level** : загруженный уровень сокращенной текстуры (mipmap), для обычной текстуры равен 0
- **internalFormat**: сколько компонентов цвета на тексель нужно записывать.
- **width**, **height** : размеры текстуры, должны быть степенями 2
- **border** : должен быть равен 0 для одномерной текстуры.
- **format**: формат данных пикселя. Допустимые значения: GL_RED, GL_RG, GL_RGB, GL_BGR, GL_RGBA, GL_BGRA, GL_RED_INTEGER, GL_RG_INTEGER, GL_RGB_INTEGER, GL_BGR_INTEGER, GL_RGBA_INTEGER, GL_BGRA_INTEGER, GL_STENCIL_INDEX, GL_DEPTH_COMPONENT, GL_DEPTH_STENCIL.
- **type** : тип данных пикселя. Допустимые значения: GL_UNSIGNED_BYTE, GL_BYTE, GL_UNSIGNED_SHORT, GL_SHORT, GL_UNSIGNED_INT, GL_INT, GL_FLOAT, GL_UNSIGNED_BYTE_3_3_2, GL_UNSIGNED_BYTE_2_3_3_REV, GL_UNSIGNED_SHORT_5_6_5, GL_UNSIGNED_SHORT_5_6_5_REV, GL_UNSIGNED_SHORT_4_4_4_4, GL_UNSIGNED_SHORT_4_4_4_4_REV, GL_UNSIGNED_SHORT_5_5_5_1, GL_UNSIGNED_SHORT_1_5_5_5_REV, GL_UNSIGNED_INT_8_8_8_8, GL_UNSIGNED_INT_8_8_8_8_REV, GL_UNSIGNED_INT_10_10_10_2, and GL_UNSIGNED_INT_2_10_10_10_REV.
- **data**: указатель на данные изображения в памяти

Текстурные координаты

- Информация о том, как наложить текстуру на геометрический объект
- Для этого задаются *текстурные координаты* каждой вершины в диапазоне от 0 до 1
- void glTexCoord2f(GLfloat s, GLfloat t);
- текстурные координаты надо задать ДО координат вершины





Наложение текстуры

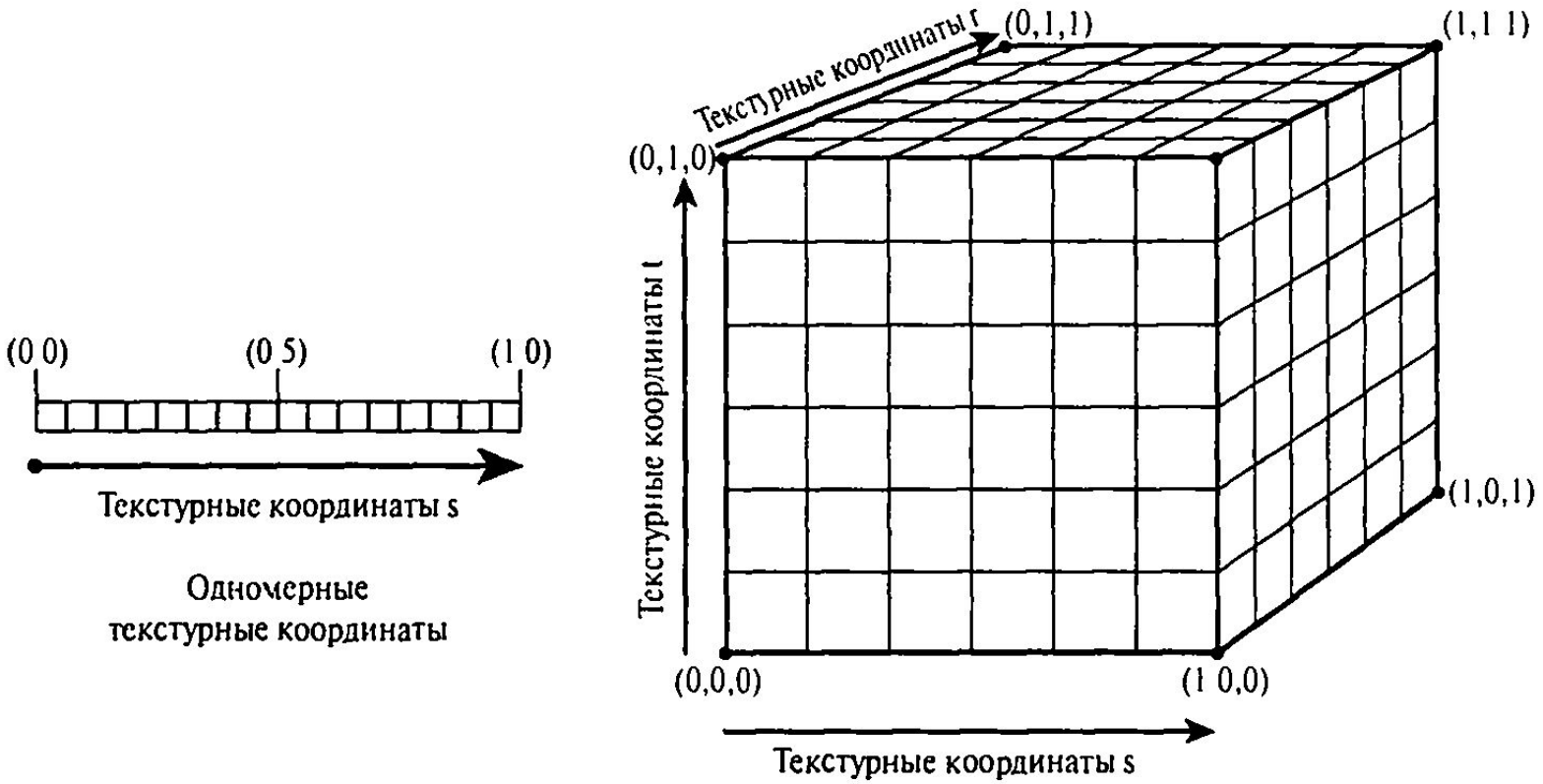
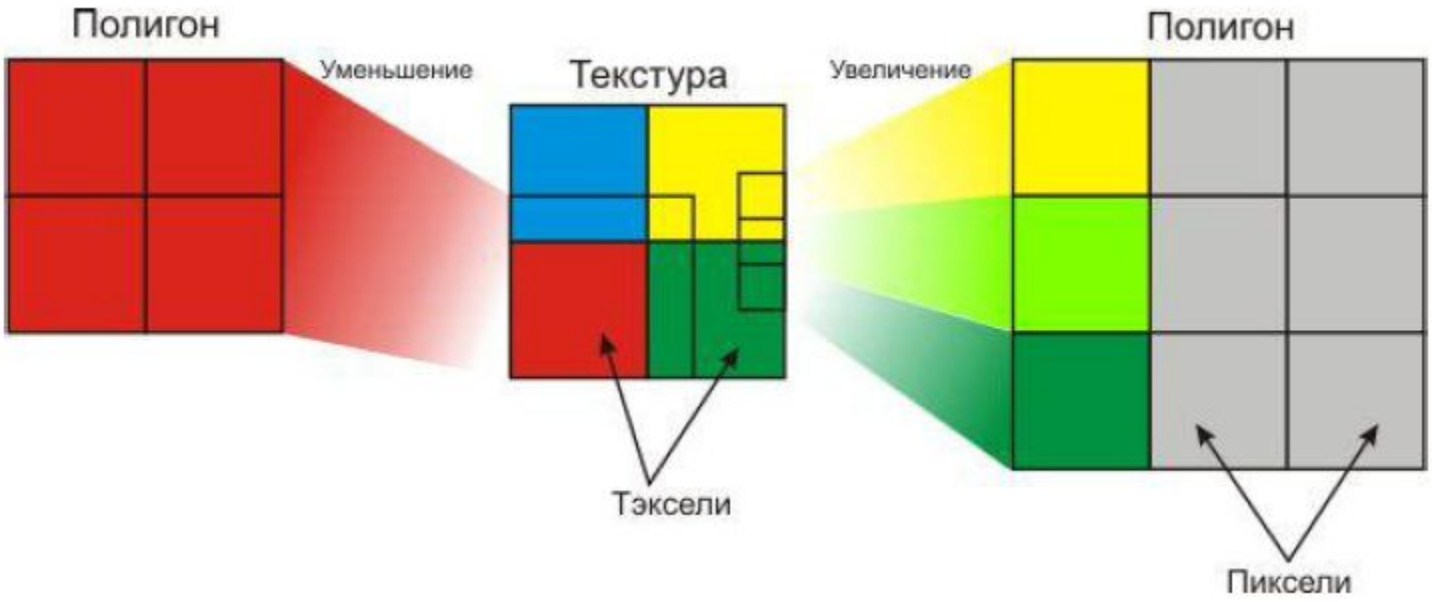


Рис. 8.2. Адресация текселей с помощью текстурных координат

Вычисление итогового цвета

- То, как OpenGL объединяет цвета текселей с цветом геометрического объекта, на который накладывается текстура, зависит от режима текстурной среды.
- void glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GLfloat param);
- param:
  - GL_ADD складывает цвет текстуры и цвет текселя
  - GL_MODULATE цвет текселя умножается на цвет геометрического объекта
  - GL_DECAL аналогично GL_REPLACE для текстур без прозрачности
  - GL_BLEND смешивание
  - GL_REPLACE цвет текселя заменяет цвет объекта
  - GL_COMBINE

Фильтрация текстуры



Фильтрация текстуры

- Между текселями и пикселами почти никогда не бывает взаимно-однозначного соответствия
- процесс расчета растянутой или сжатой картой текстуры называется *фильтрацией*
- void glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GLfloat param);
  - GL_NEAREST
  - GL_LINEAR
  - GL_NEAREST_MIPMAP_NEAREST
  - GL_LINEAR_MIPMAP_NEAREST
  - GL_NEAREST_MIPMAP_LINEAR
  - GL_LINEAR_MIPMAP_LINEAR

Фильтрация текстуры

- void glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GLfloat param);
  - GL_NEAREST
  - GL_LINEAR

Намотка текстуры

- Если текстурные координаты выходят из диапазона [0, 1], то OpenGL обрабатывает их согласно текущему режиму *намотки,* его можно установить отдельно для каждой координаты
- void glTexParameteri(GL_TEXTURE_2D GL_TEXTURE_WRAP_S , GLfloat param);
- void glTexParameteri(GL_TEXTURE_2D GL_TEXTURE_WRAP_T , GLfloat param);
  - GL_CLAMP_TO_EDGE
  - GL_MIRRORED_REPEAT
  - GL_REPEAT

Текстурные объекты

- Переиспользование загруженной текстуры
  - Сгенерируйте имена текстур.
  - Привяжите объекты текстуры к данным текстуры (в частности к массивам изображений и свойствам).
  - Выбирайте (повторно связывайте) ваши текстурные объекты, делая их текущими для визуализации текстурированных моделей.

Именование текстурных объектов

- В качестве имени текстуры может быть использовано любое ненулевое беззнаковое целое. Для получения неиспользуемых имен текстур используйте **glGenTextures()**.
- void **glGenTextures (GLsizei *n, GLuint *textureNmes);***
- **glBindTexture()** используется и при создании, и при использовании текстурных объектов.
  - При начальном связывании создается новый текстурный объект.
  - Когда объект текстуры связывается впоследствии, данные, содержащиеся в нем, становятся текущим состоянием текстуры, замещая предыдущее состояние.

Очистка текстурных объектов

- void **glDeleteTextures (GLsizei *n, const GLuint *textureNames);***
- Удаляет *n текстурных объектов, чьи имена переданы в аргументе* textureNames. Освобожденные имена могут быть повторно использованы **Если удаляется текущая текстура, состояние переходит к** текстуре по умолчанию с *textureName* равным 0. Попытки удаления имен несуществующих текстур игнорируются без генерации каких-либо ошибок.

Загрузка текстур в OpenTK

```
using System.Drawing;
using System.Drawing.Imaging;
using OpenTK.Graphics.OpenGL;
static int LoadTexture(string filename)
{
    if (String.IsNullOrEmpty(filename))
      throw new ArgumentException(filename);
    int id = GL.GenTexture();
    GL.BindTexture(TextureTarget.Texture2D, id);
    GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMinFilter,
      (int)TextureMinFilter.Linear);
    GL.TexParameter(TextureTarget.Texture2D, TextureParameterName.TextureMagFilter,
      (int)TextureMagFilter.Linear);
    Bitmap bmp = new Bitmap(filename);
    BitmapData bmp_data = bmp.LockBits(new Rectangle(0, 0, bmp.Width, bmp.Height),
    ImageLockMode.ReadOnly,    System.Drawing.Imaging.PixelFormat.Format32bppArgb);
    GL.TexImage2D(TextureTarget.Texture2D, 0, PixelInternalFormat.Rgba, bmp_data.Width, bmp_data.Height,
      0, OpenTK.Graphics.OpenGL.PixelFormat.Bgra, PixelType.UnsignedByte, bmp_data.Scan0);
    bmp.UnlockBits(bmp_data);
    return id;
}
```

Автоматическое генерирование текстурных координат

- void **glTexGen{ifd} (GLenum *coord, GLenum pname, TYPE param);***
- void **glTexGen{ifd}v (GLenum *coord, GLenum pname, TYPE *param);***
  - *coord: GL_S, GL_T, GL_R*
  - *pname* ***может*** принимать значения GL_TEXTURE_GEN_MODE, GL_OBJECT_PLANE, GL_EYE_PLANE, GL_SPHERE_MAP. Если задано значение GL_TEXTURE_GEN_MODE, *param* ***должен быть*** целым числом (или указателем на целое число, если используется векторная версия команды), которое равно GL_OBJECT_LINEAR, GL_EYE_LINEAR или GL_SPHERE_MAP.
  - Эти символические константы указывают на то, какая функция должна использоваться для вычисления координат текстуры. Со всеми остальными возможными значениями для *pname, param* ***должен представлять собой указатель на массив величин*** (***в векторной*** версии команды), задавая параметры функции вычисления текстурных координат.

Mipmapping

- Сокращенные или множественные тукстуры
- Серии предварительно отфильтрованных карт текстуры с разными разрешениями (128x128, 64x64, 32x32 и так далее)