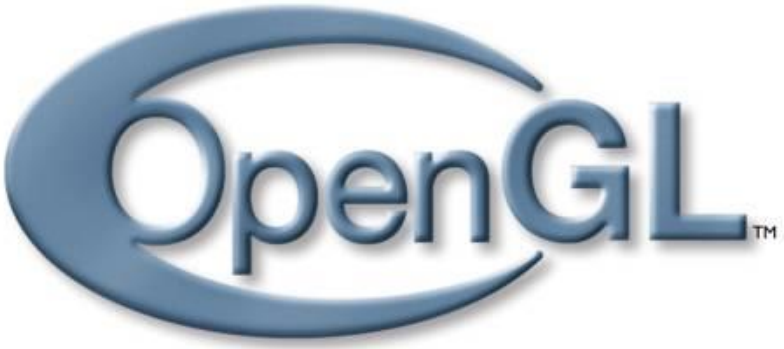
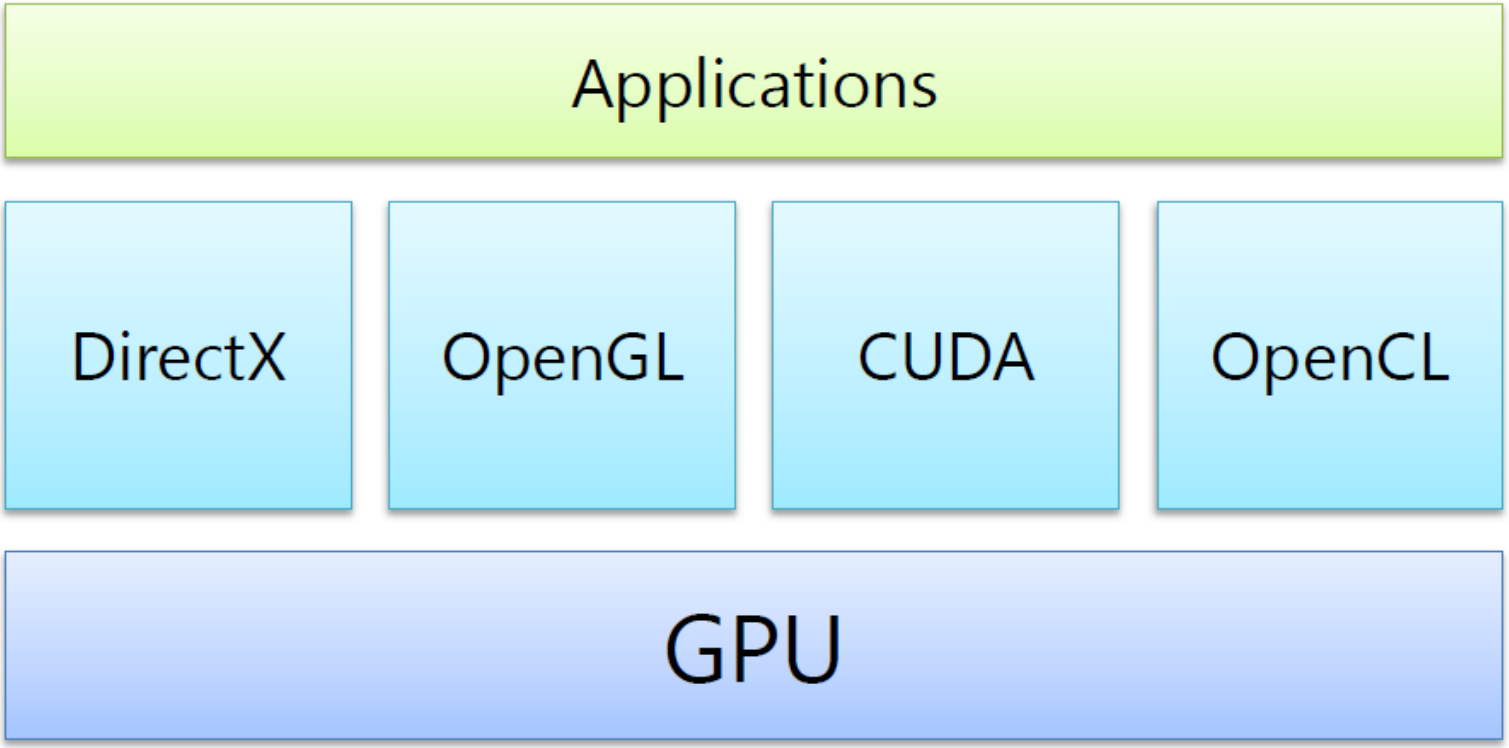
Программный интерфейс OpenGL 4

Владимир Фролов (vfrolov@graphics.cs.msu.ru)
И.П.М. им. Келдыша, ВМиК МГУ

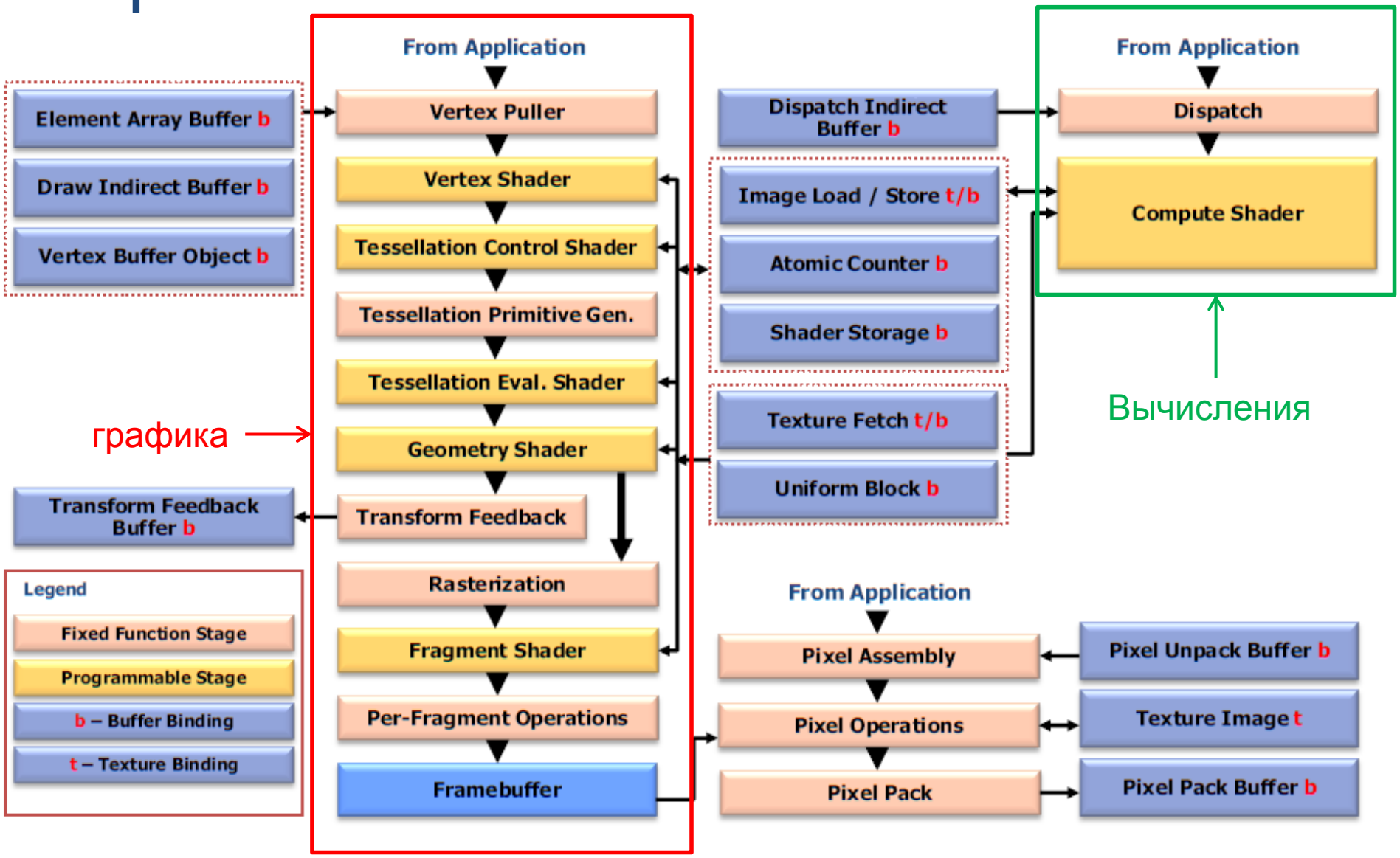


API программирования GPU

- Графика
- Вычисления общего назначения

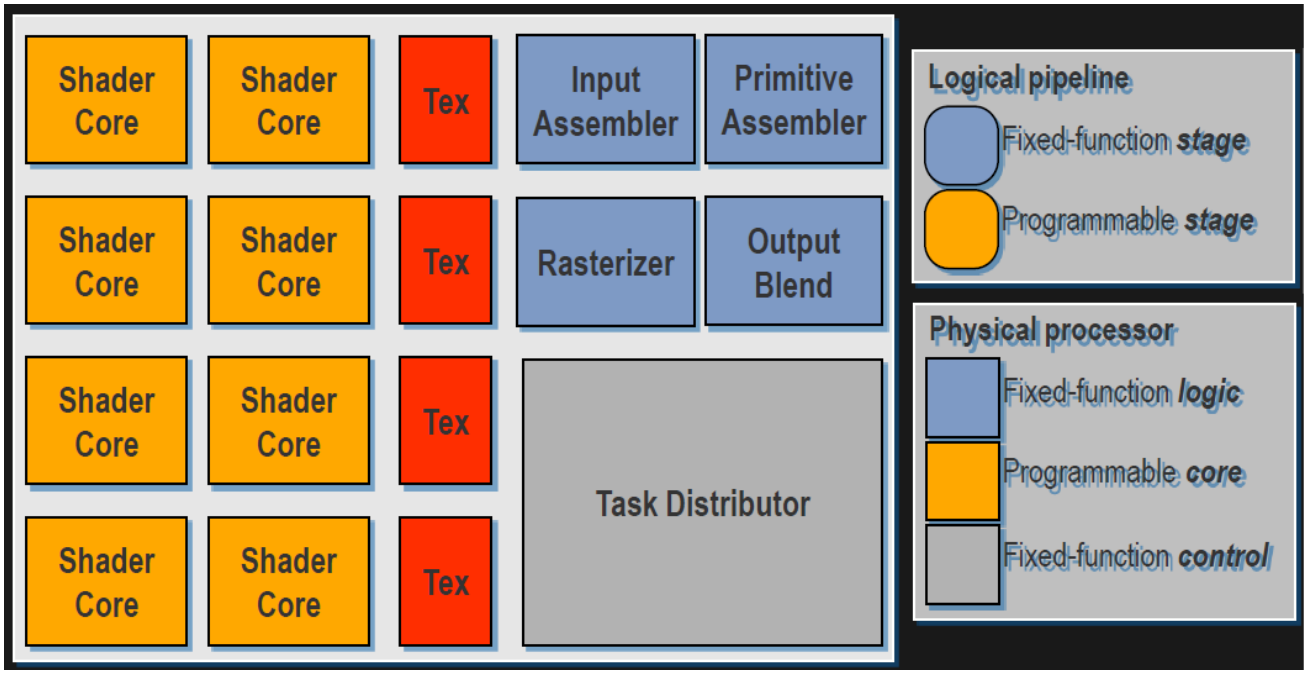


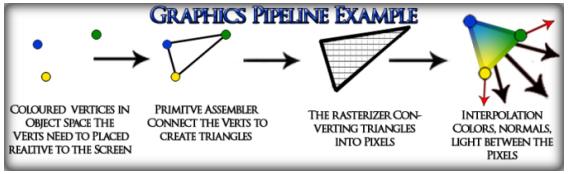
OpenGL 4.3

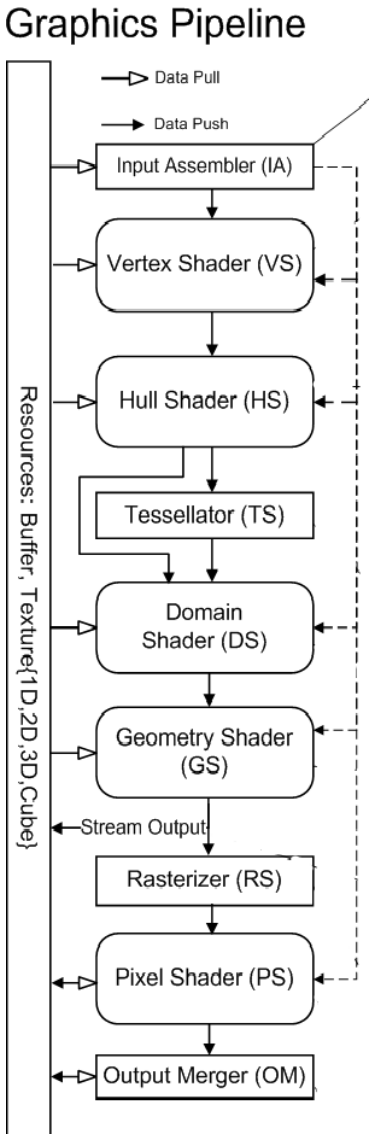

From Application
Element Array Buffer b
Draw Indirect Buffer b
Vertex Buffer Object b
Vertex Puller
Vertex Shader
Tessellation Control Shader
Tessellation Primitive Gen.
Tessellation Eval. Shader
Geometry Shader
Transform Feedback
Rasterization
Fragment Shader
Per-Fragment Operations
Framebuffer
графика
Transform Feedback Buffer b
Legend
Fixed Function Stage
Programmable Stage
b – Buffer Binding
t – Texture Binding
Dispatch Indirect Buffer b
Image Load / Store t/b
Atomic Counter b
Shader Storage b
Texture Fetch t/b
Uniform Block b
From Application
Dispatch
Compute Shader
Вычисления
From Application
Pixel Assembly
Pixel Operations
Pixel Pack
Pixel Unpack Buffer b
Texture Image t
Pixel Pack Buffer b

Графические процесоры

- Графика
- Вычисления общего назначения





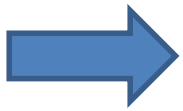


Как выбрать API?

- На ОС какого типа должна работать моя программа?
  - Windows => GL || DX
  - Linux/Android => GL (GL/ES)
  - MacOS => GL
- На каком железе должна работать моя программа?
  - DX11 hardware => GL 4+ || DX11
  - DX10 hardware => GL 3+ || DX11
  - DX9 hardware => GL 2+ || DX11 || DX9 (WinXP)
  - Везде => GL 1.0 || DX11 (WARP device) ислючая XP

OpenGL3 и OpenGL4

- Разница достаточно тонкая
  - Sampler Objects
  - Separate Shader Objects
  - Program Pipeline
  - Transform Feedback Object
  - …



Сближает GL4 с DX11

  - Новые фитчи:
    - Вычислительные шейдеры
      - GL_SHADER_STORAGE_BUFFER
      - glClearBufferObject (как memset в Си)
      - ..
    - Тесселяция
    - double (64 битная плавающая точка)
    - Image load/store
    - Атомарные операции
    - …

OpenGL1

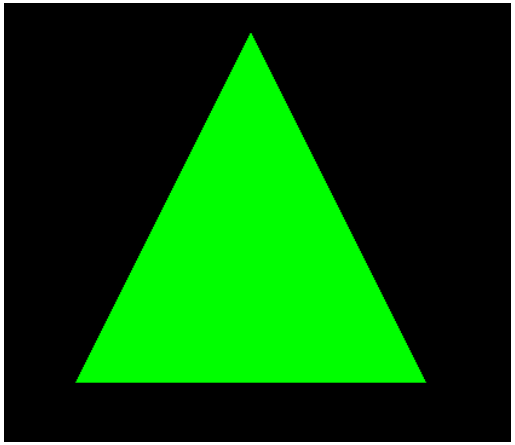
- Установкка матриц

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0, windowWidth, 0, windowHeight);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.5, 0.5, 0.0);
glScalef(0.5, 0.5, 0.0);
glScalef(0.5, 0.5, 0.0);
glRotatef(angle, 0, 0, 1);
```

…..

- Рисование

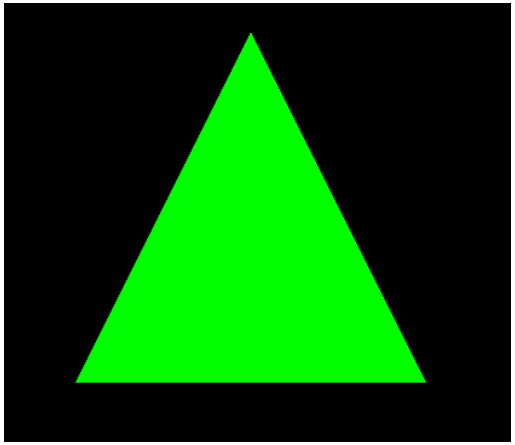
```
glColor3f(0.0, 1.0, 0.0);
glBegin(GL_TRIANGLES);
glVertex3f(-0.5, +0.5, 0.0);
glVertex3f(+0.5, -0.5, 0.0);
glVertex3f(+0.0, +0.5, 0.0);
glEnd();
```



OpenGL2

- CPU

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.5, 0.5, 0.0);
glScalef(0.5, 0.5, 0.0);
glScalef(0.5, 0.5, 0.0);
glRotatef(angle, 1, 0, 0);

glBegin(GL_TRIANGLES);
glVertex3f(-0.5, 0.5, 0.0);
glVertex3f(+0.5, 0.5, 0.0);
glVertex3f(+0.0, 0.5, 0.0);
glEnd();
```



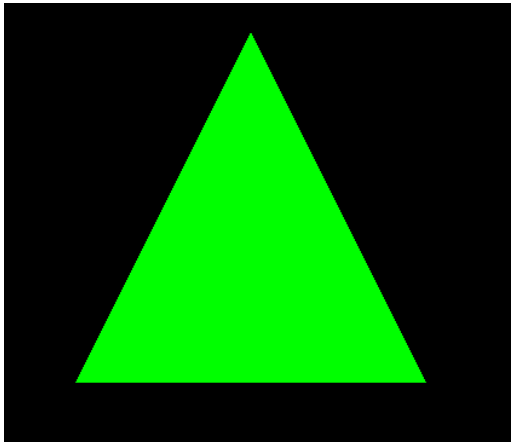
- Шейдеры (GPU)

```
void main(void)
{
  gl_Position = gl_ModelViewMatrix*(gl_ProjectionMatrix*gl_Vertex);
  // gl_Position = ftransform();
}

void main(void)
{
  gl_FragColor = vec4(0.0, 1.0, 0.0, 1.0);
}
```

OpenGL3

- CPU

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(0.5, 0.5, 0.0);
glScalef(0.5, 0.5, 0.0);
glScalef(0.5, 0.5, 0.0);
glRotatef(angle, 1, 0, 0);

glBegin(GL_TRIANGLES);
glVertex3f(-0.5, 0.5, 0.0);
glVertex3f(+0.5, 0.5, 0.0);
glVertex3f(+0.0, 0.5, 0.0);
glEnd();
```

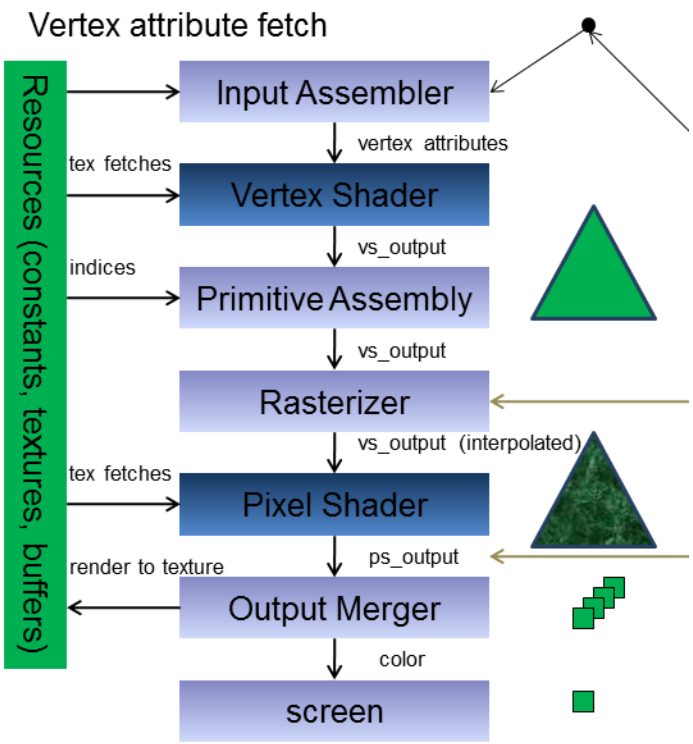


- Шейдеры (GPU)

```
void main(void)
{
  gl_Position = gl_ModelViewMatrix*(gl_ProjectionMatrix*gl_Vertex);
  // gl_Position = ftransform();
}

void main(void)
{
  gl_FragColor = vec4(0.0, 1.0, 0.0, 1.0);
}
```

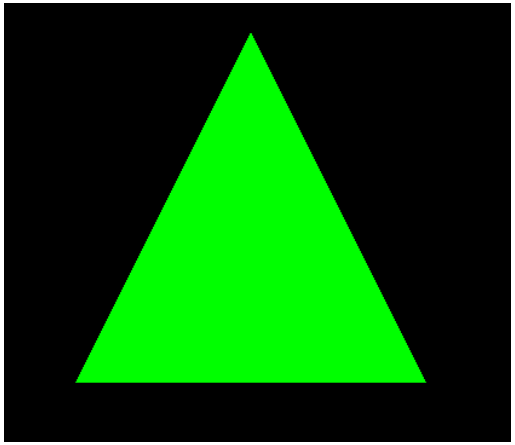
OpenGL3

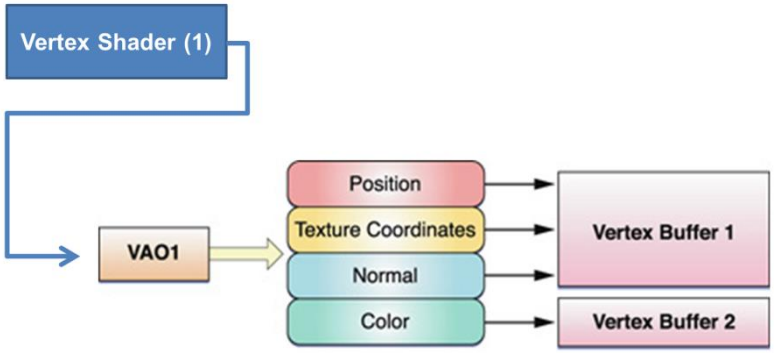
- Работы стало больше
  - Загрузка текста шейдеров из файла
  - Компиляция (vs_text, ps_text) -> (vs, fs)
  - Линковка (vs, fs) => program
    - g_prog = ShaderProgram(…);
  - Создание и загрузка данных
    - GLuint g_vbo1 = … // загружаем позиции вершин
    - GLuint g_vbo2 = … // на GPU
    - (g_vbo1, g_vbo2) => g_vao
  - Привязка ресурсов, задание констант
    - glUniformMatrix4fv(…);
    - bindTexure(…)
  - Указать VS откуда читать данные
    - glBindVertexArray(g_vao);
  - Draw call
    - glDrawArrays(GL_TRIANGLES, 0, 3); // 3 – кол-во вызовов вершинного шейдера



OpenGL3 (1 - VBO, VAO)

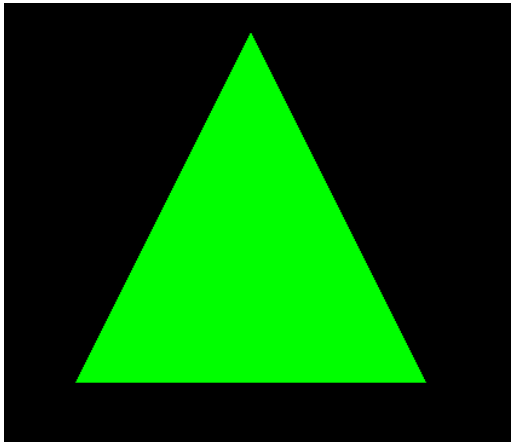
```
float trianglePos[] =  {
     -0.5f, -0.5f,
      0.5f, -0.5f,
      0.0f, +0.5f,
};
GLuint g_vertexBufferObject = 0;
GLuint g_vertexArrayObject  = 0;
GLuint vertexLocation = 0;

glGenBuffers(1, &g_vertexBufferObject);
glBindBuffer(GL_ARRAY_BUFFER, g_vertexBufferObject);
glBufferData(GL_ARRAY_BUFFER, 3*2*sizeof(GLfloat), (GLfloat*)trianglePos, GL_STATIC_DRAW);

glGenVertexArrays(1, &g_vertexArrayObject);
glBindVertexArray(g_vertexArrayObject);

glBindBuffer(GL_ARRAY_BUFFER, g_vertexBufferObject);
glEnableVertexAttribArray(vertexLocation);
glVertexAttribPointer(vertexLocation, 2, GL_FLOAT, GL_FALS
```





OpenGL3 (2 – Shader setup and Draw)

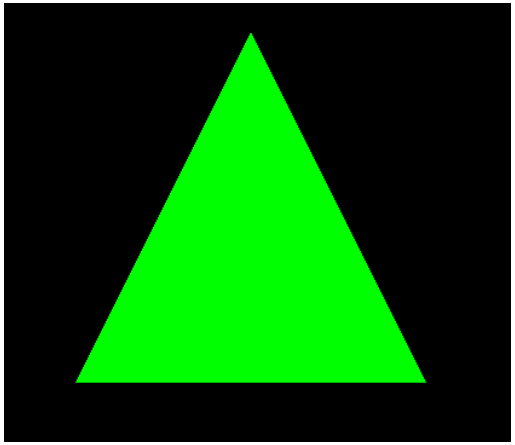
```
float model[16];
float modelView[16];

glusLoadIdentityf(model);
glusRotateRzRyRxf(model, input.cam_rot[0], input.cam_rot[1], 0.0f);
glusLookAtf(modelView, 0.0f, 0.0f, -2.0f,
                       0.0f, 0.0f, 0.0f,
                       0.0f, 1.0f, 0.0f);

glusMultMatrixf(modelView, modelView, model);
```

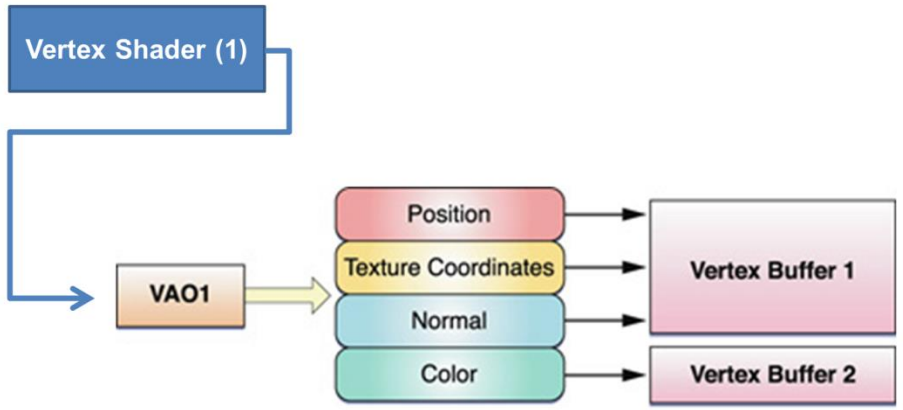


```
GLint location = glGetUniformLocation(g_program.program, "modelViewMatrix");
if(location >= 0)
  glUniformMatrix4fv(location, 1, GL_FALSE, (GLfloat*)&modelView);

... // same for projection matrix

glBindVertexArray(g_vertexArrayObject);
glDrawArrays(GL_TRIANGLES, 0, 3);
```

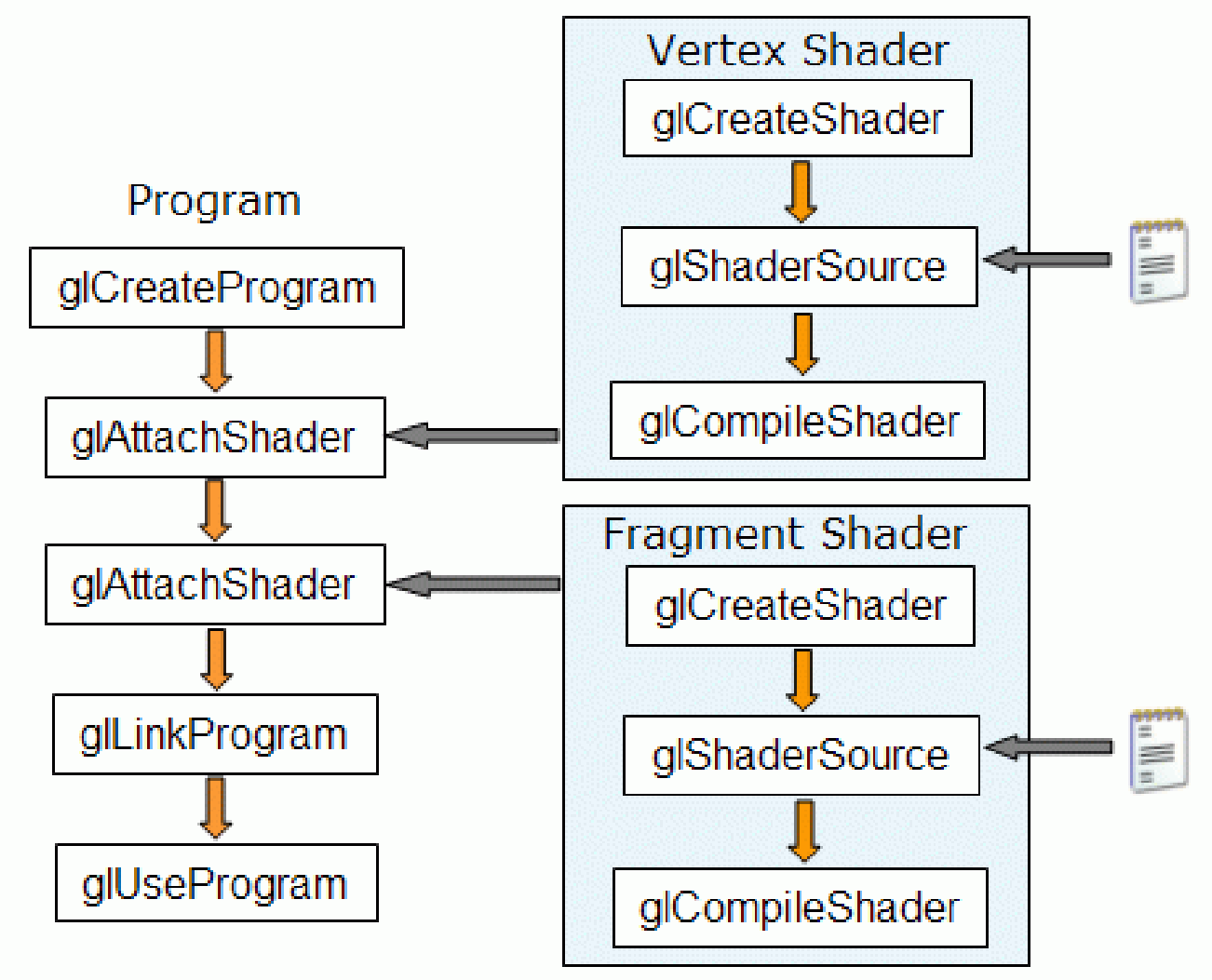
OpenGL3 (3 –Shaders)

```
// vertex shader
#version 330

uniform mat4 projectionMatrix;
uniform mat4 modelViewMatrix;

in  vec2 vertex;

void main(void)
{
  vec4 viewPos = modelViewMatrix*vec4(vertex,0.0,1.0);
  gl_Position     = projectionMatrix*viewPos;
}

// fragment shader
#version 330
out vec4 fragColor;

void main(void)
{
  fragColor = vec4(0,1,0,0);
}
```



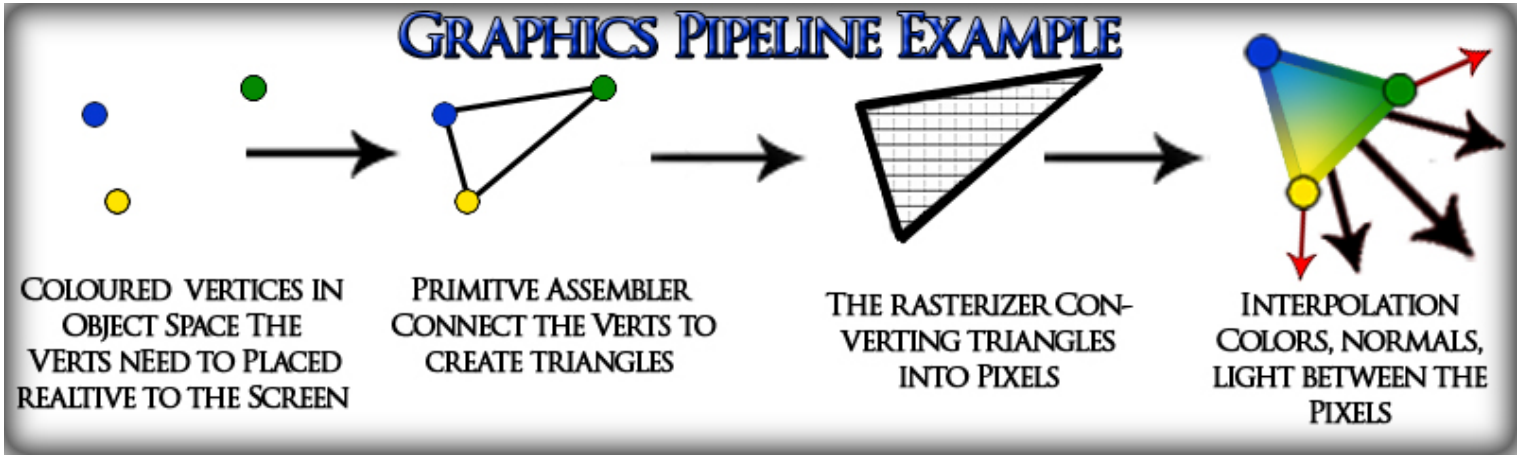


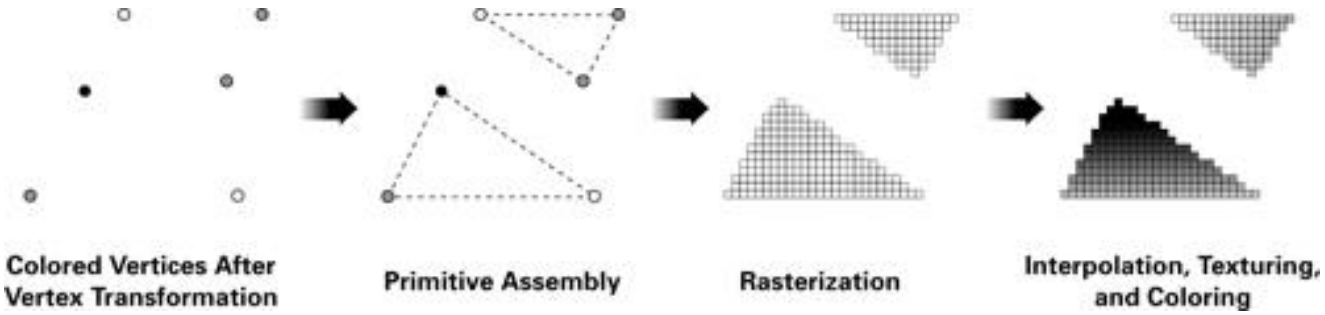
Создание объекта программы



Графический конвейер

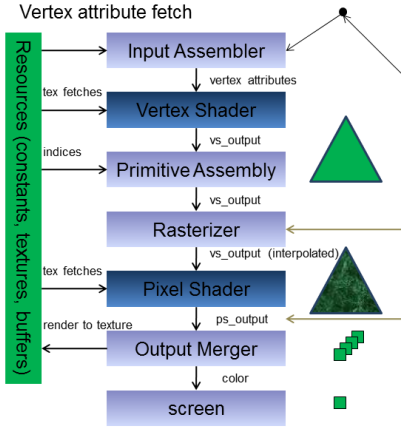
- 3D сцена => 2D изображение
- Некоторые стадии фиксированные, некоторые программируемые

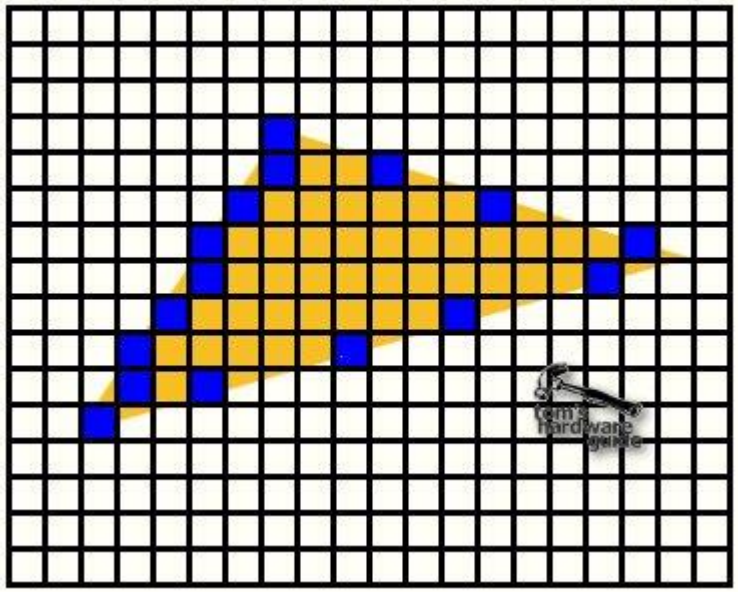


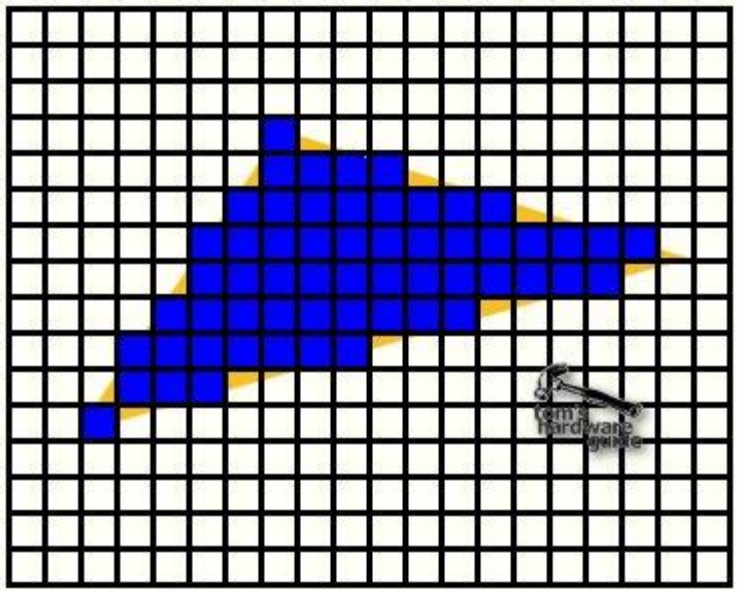


Растеризатор

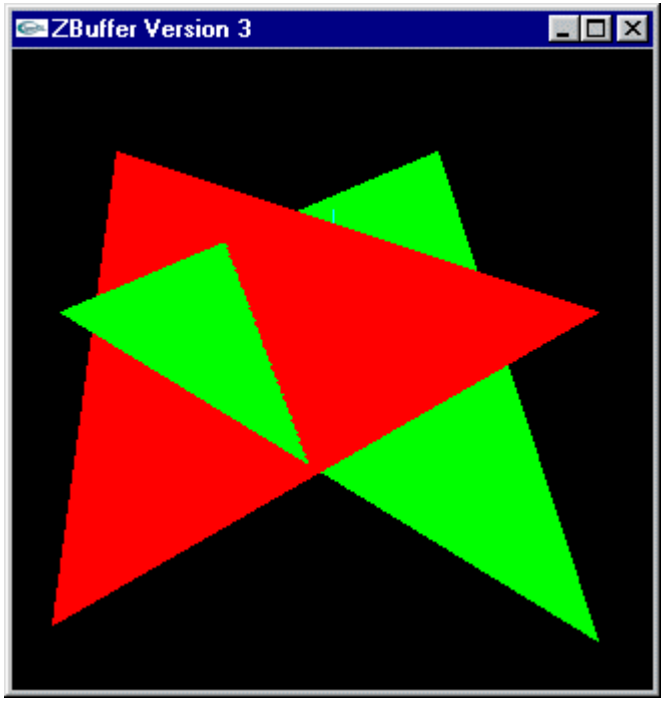
- Отсечение
- Растеризация
- Z-буфер (Early-z test)
- Буфер трафарета (Stencil test)
- Интерполяция атрибутов





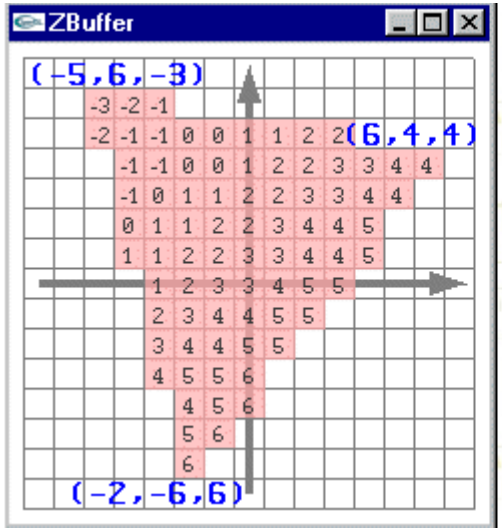


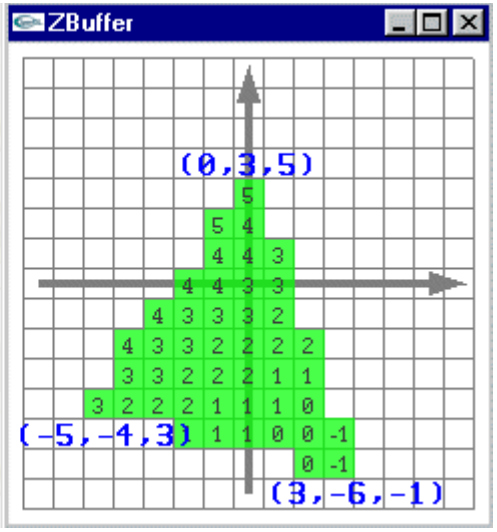
Z-Buffer

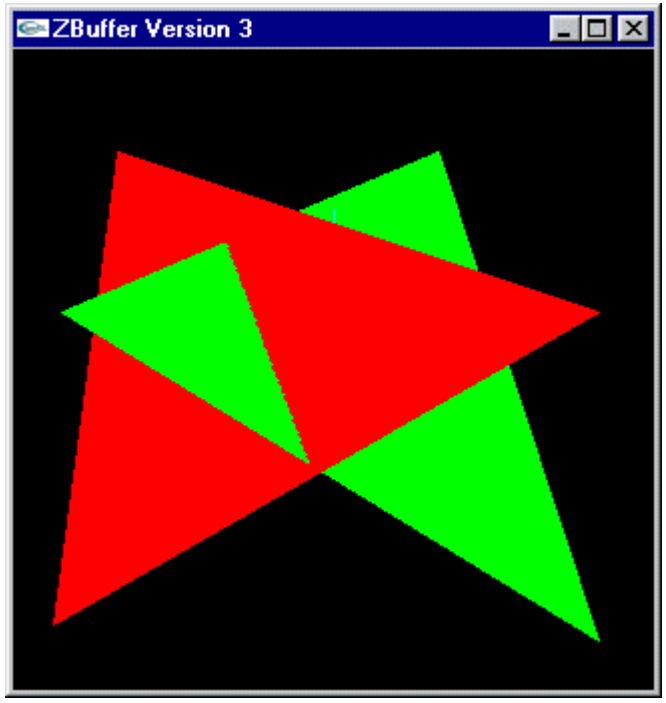


http://raphaello.univ-fcomte.fr/ig/zbuffer/zbuffer.htm

Z-Buffer

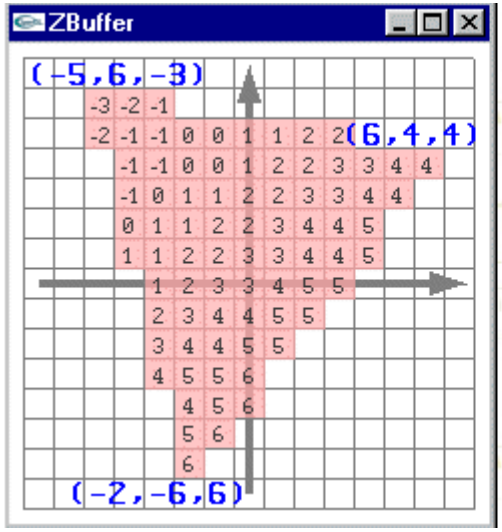


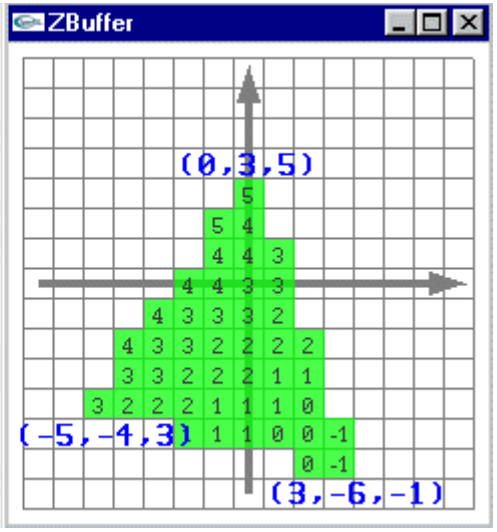


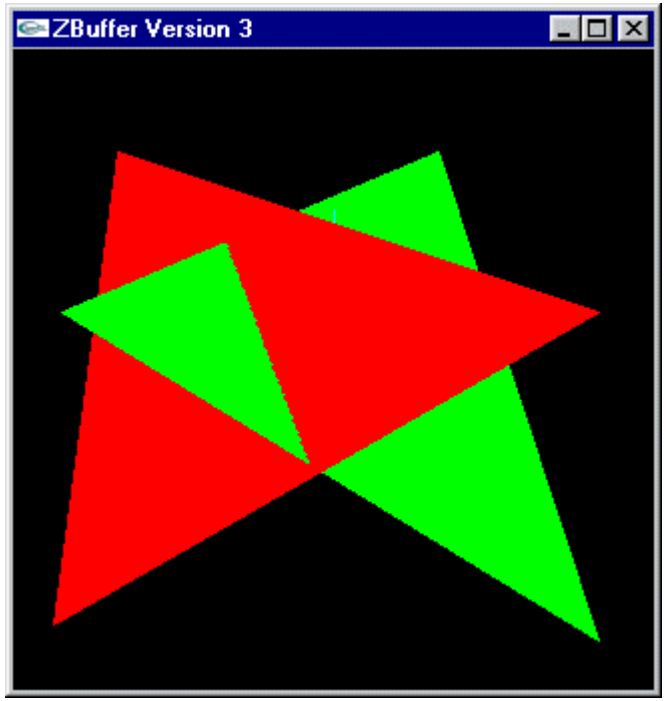


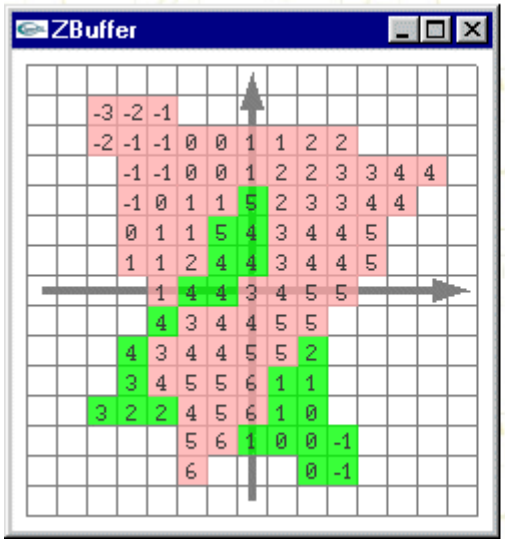
http://raphaello.univ-fcomte.fr/ig/zbuffer/zbuffer.htm

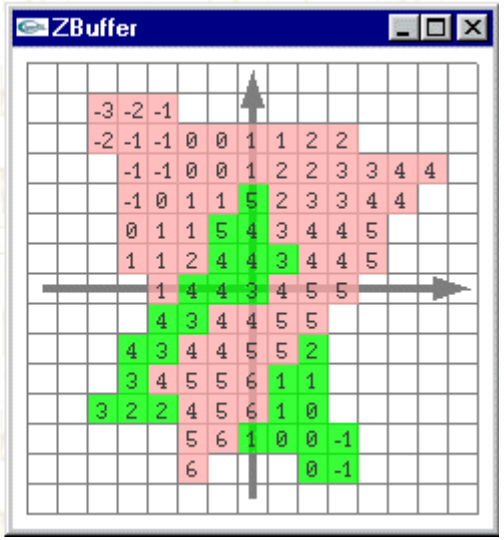
Z-Buffer









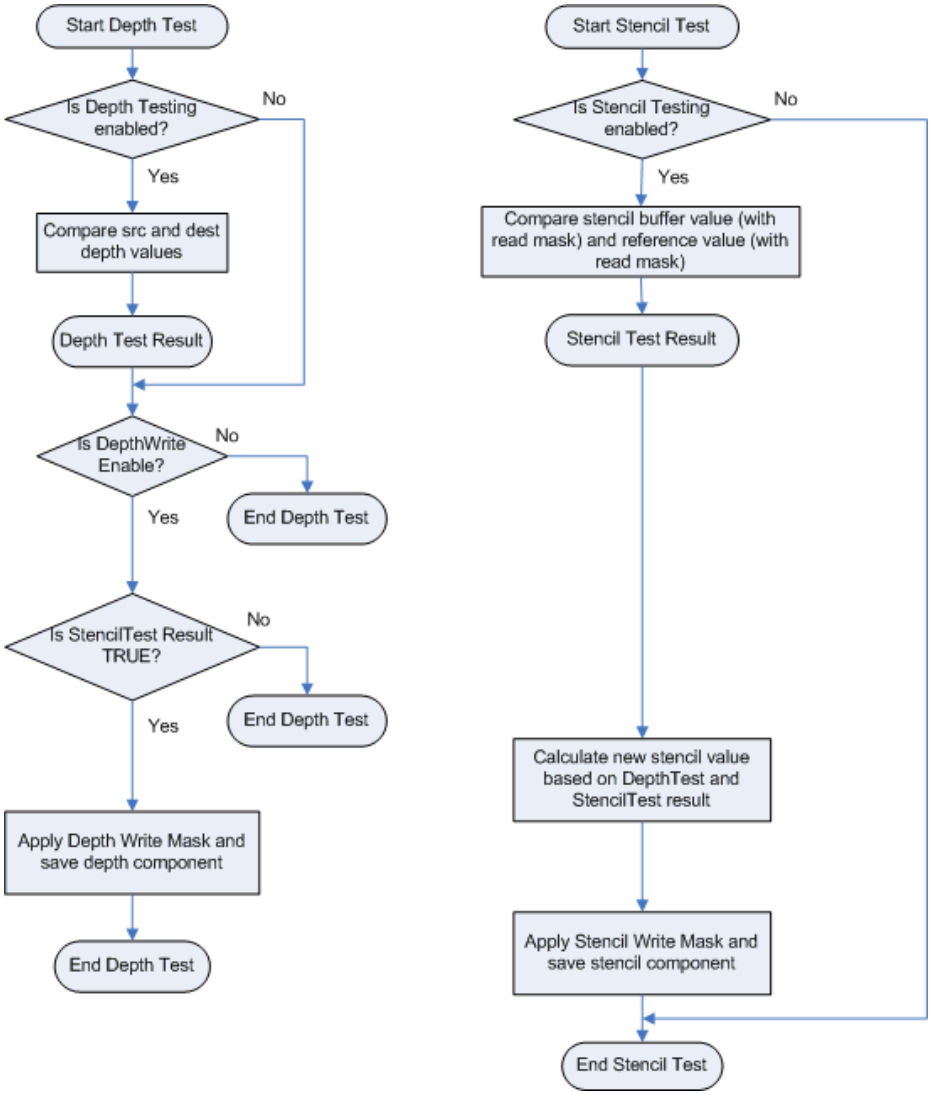


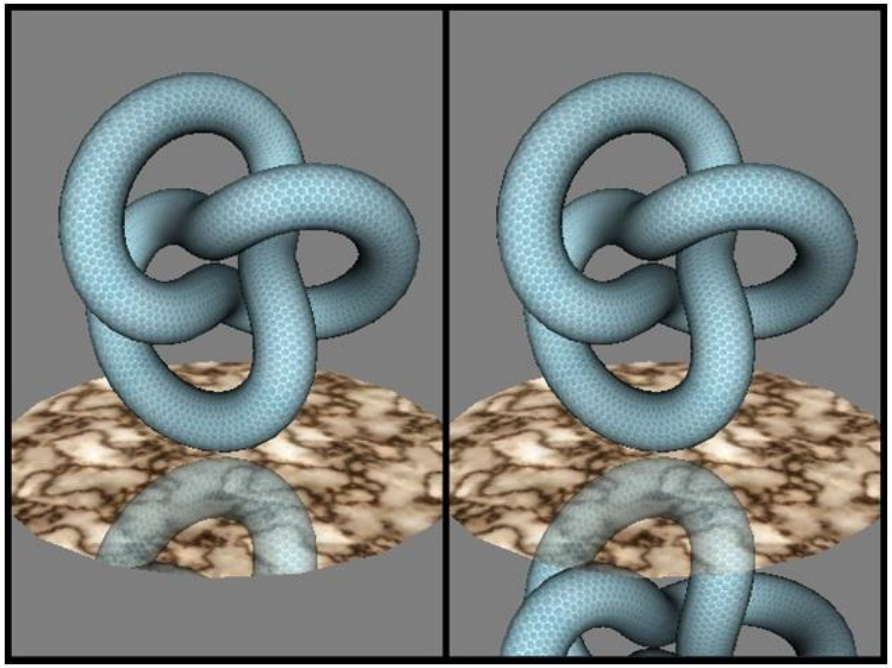
http://raphaello.univ-fcomte.fr/ig/zbuffer/zbuffer.htm

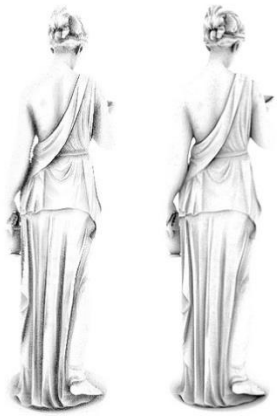
Z-buffer fighting

- Как избежать?
  - Сделать больше точность (доступно 16, 24 или 32 бит)
  - Не рисовать геометрию на одной и той же глубине
  - Определить порядок отрисовки и использовать `glDepthFunc(GL_LESS);`

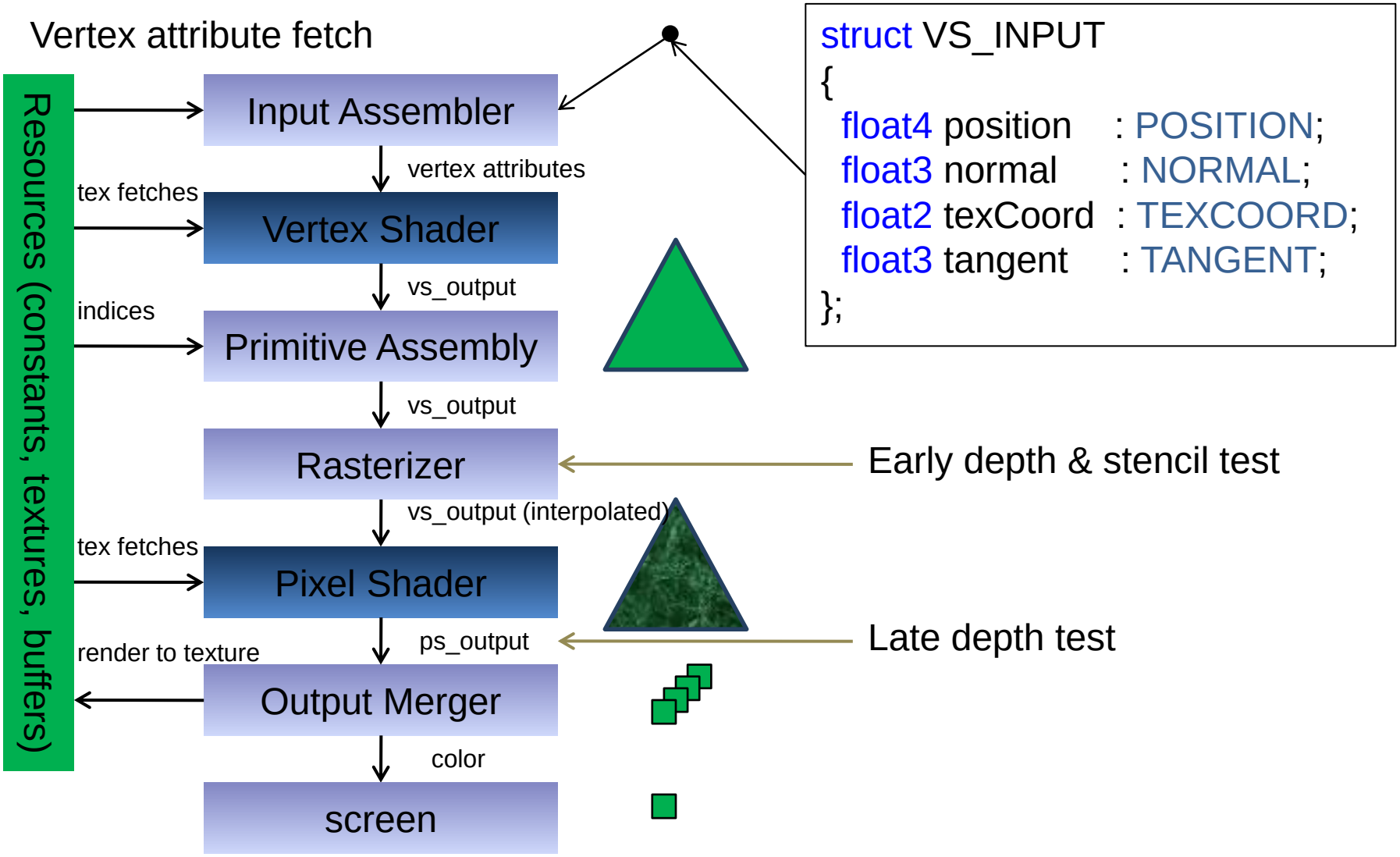
Depth & Stencil Test





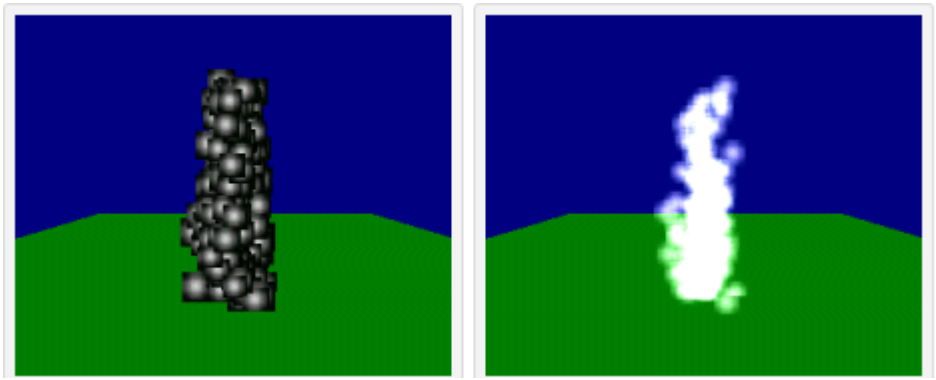


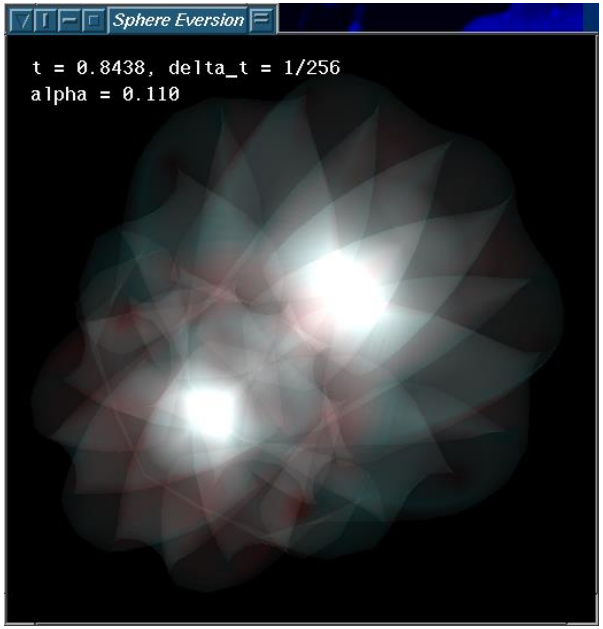
DX9 Pipeline (OpenGL 2.0, 3.0)



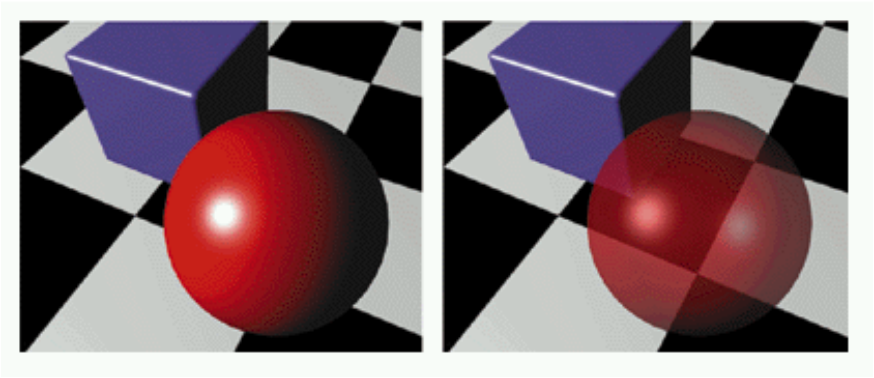
Output Merger

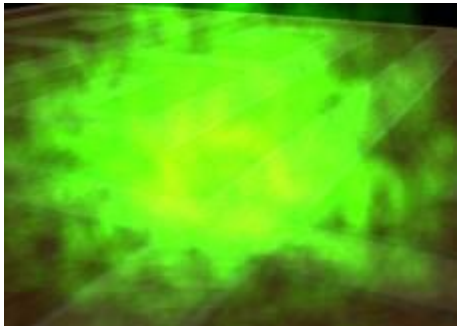
- Альфа-смешивание





From Computer Desktop Encyclopedia
Reprinted with permission.
© 1998 Intergraph Computer Systems





Output Merger

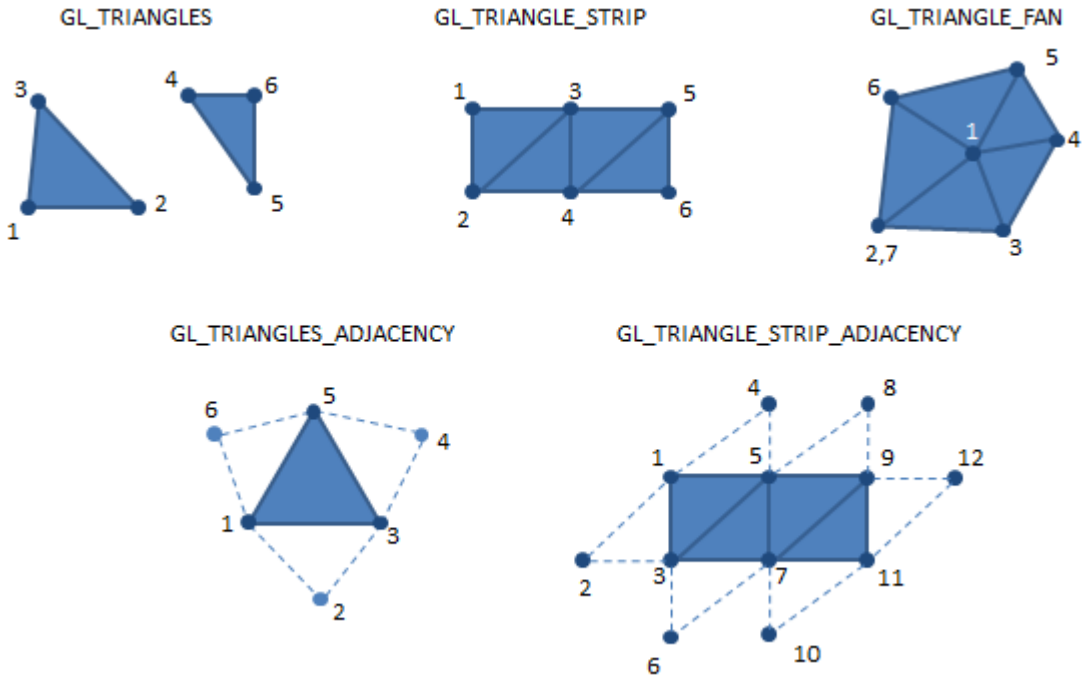
- Альфа-смешивание
  - glEnable(GL_BLEND)
  - glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
  - ...
  - glDisable();

$$C' = \alpha_F C_F + (1 - \alpha_F) C_B$$

Графический конвейер

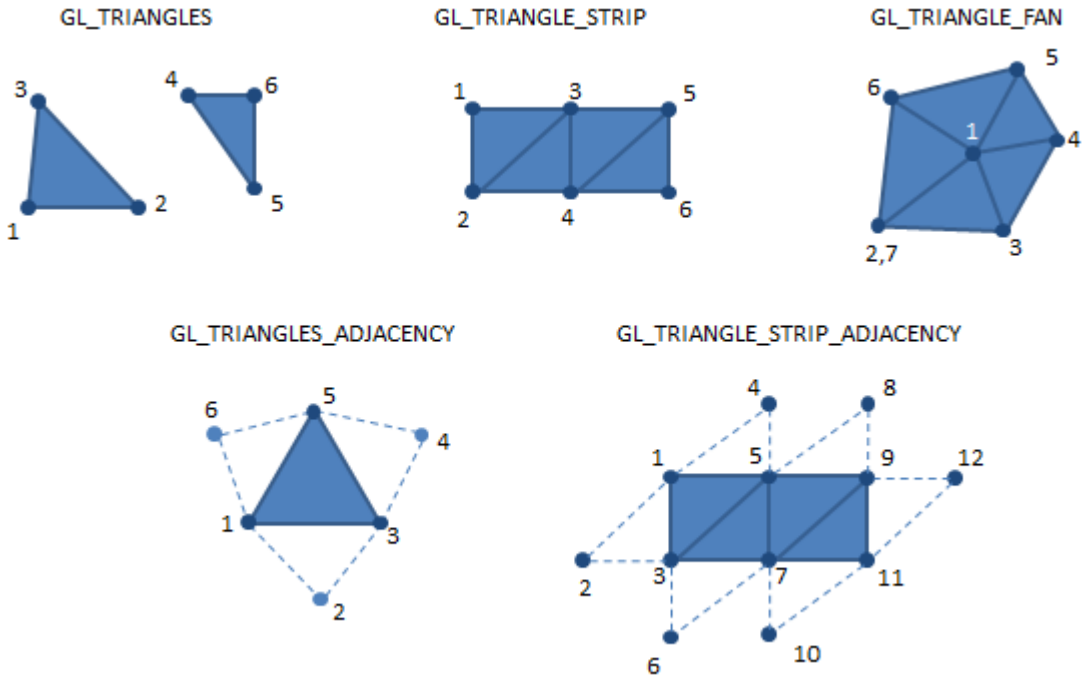
Графический конвейер

- glDrawArrays(GL_TRIANGLES, 0, ?); // 2 треугольника
- glDrawArrays(GL_TRIANGLE_STRIP 0, ?); // 2 треугольника



Графический конвейер

- glDrawArrays(GL_TRIANGLES, 0, 6); // 2 треугольника
- glDrawArrays(GL_TRIANGLE_STRIP 0, 4); // 2 треугольника



OpenGL

- API для доступа к функциональности GPU
- Мультиплатформенный
- Мультиязыковой
- Процедурный
- Динамически расширяемый
- Указатели на функции
  - wglGetProcAdress(…)
- Доступ к объектам осуществляется при помощи целочисленный имен
- Понятие текущего объекта

```
glGenBuffers (1, &m_vbo);
glBindBuffer(GL_ARRAY_BUFFER, m_vbo);
glBufferData(…);
```

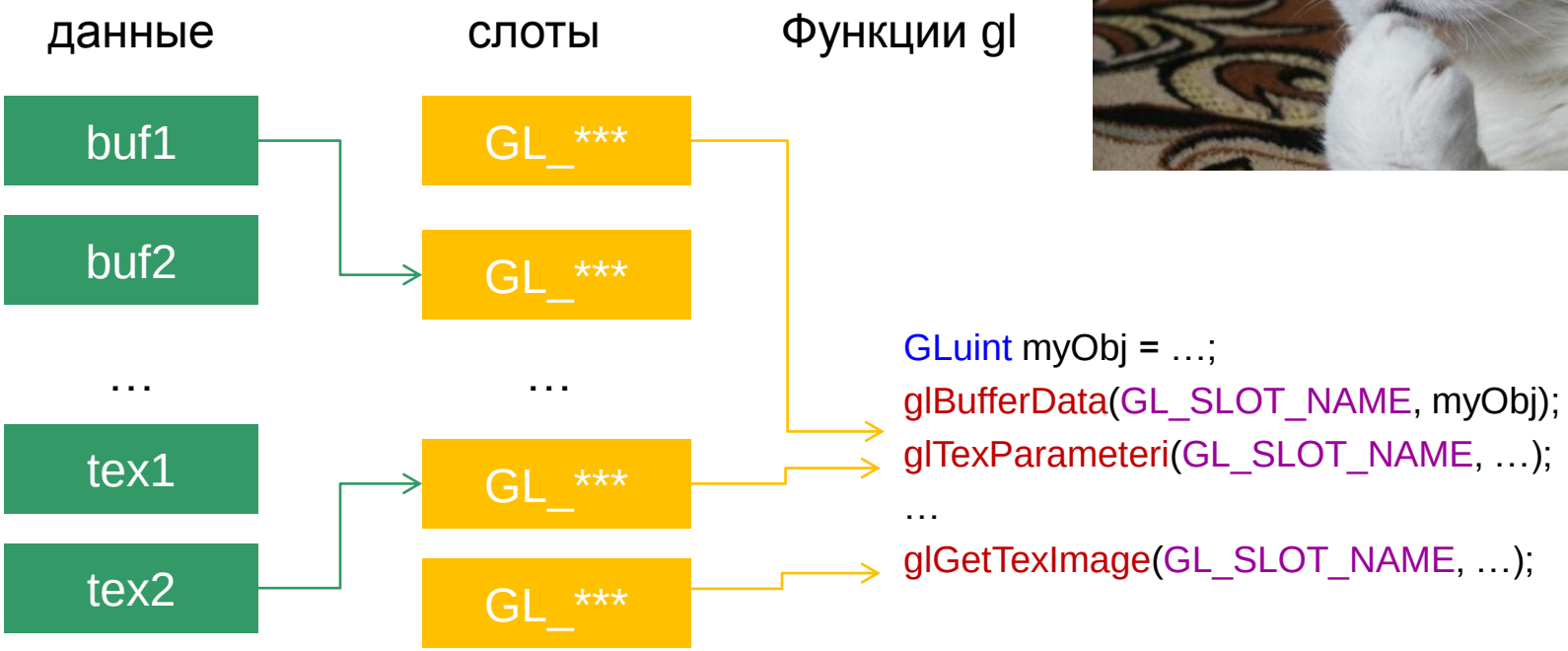
OpenGL

- Мультиязыковой ?
- Указатели на функции ?
  - wglGetProcAdress(…)
- Целочисленный имена ?
- Понятие текущего объекта ?

```
glBindBuffer(GL_ARRAY_BUFFER, m_vbo);
glBufferData(GL_ARRAY_BUFFER, …);
…
glBindTexture(GL_TEXTURE_2D, m_colorTexture);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

glBindTexture(GL_TEXTURE_2D, 0);
```

Идеология OpenGL

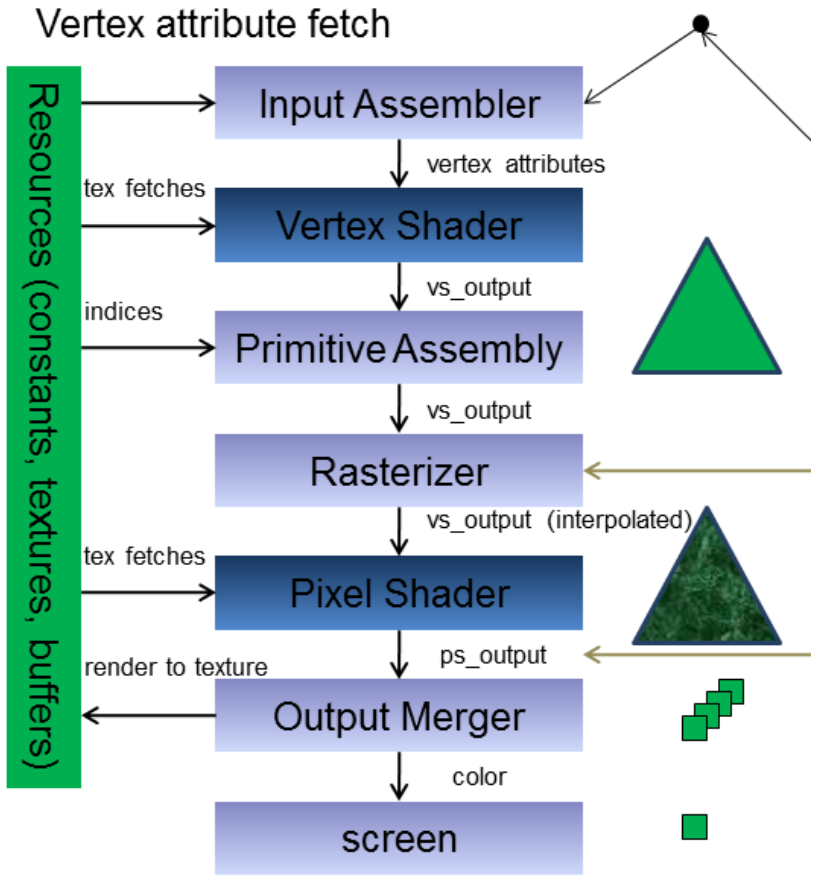
```
GLuint myObj = …;
glBindSomeObject(GL_SOME_TARGET_NAME, myObj);
glDoSomething(GL_SOME_TARGET_NAME, …);
```




```
GLuint myObj = …;
glBufferData(GL_SLOT_NAME, myObj);
glTexParameteri(GL_SLOT_NAME, …);
…
glGetTexImage(GL_SLOT_NAME, …);
```

Объекты и сущности

- Контекст
- VAO
- Шейдеры
- Константы
- Буферы (VBO, PBO, UBO …)
- Текстуры
- FBO
- Sampler (OpenGL 4)
- *Image (OpenGL 4)
- Состояния
- Специальные объекты (много в GL4)
  - (типа Transform Feedback-Object, program-pipeline)



OpenGL buffer object

- Создание буфера

```
glGenBuffers(1, &vbo);
```

- Работа с буфером

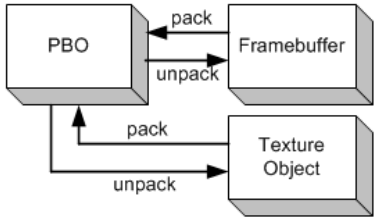
```
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, N*sizeof(GLfloat), (GLfloat*)cpuData, GL_STATIC_DRAW);
```

- Семантики буферов (‘target’)
  - GL_ARRAY_BUFFER - буфер для вершин (VBO)
  - GL_ATOMIC_COUNTER_BUFFER - для атомарных операций
  - GL_COPY_READ_BUFFER - копирование типа буфер => буфер
  - GL_COPY_WRITE_BUFFER - копирование типа буфер => буфер
  - GL_DRAW_INDIRECT_BUFFER - для “draw indirect”
  - GL_DISPATCH_INDIRECT_BUFFER - для “dispatch indirect”
  - GL_ELEMENT_ARRAY_BUFFER - буфер индексов
  - GL_PIXEL_PACK_BUFFER - copy (текстура => буфер) (PBO)
  - GL_PIXEL_UNPACK_BUFFER - copy (буфер => текстура) (PBO)
  - GL_SHADER_STORAGE_BUFFER - для compute shader-a
  - GL_TEXTURE_BUFFER - TBO
  - GL_TRANSFORM_FEEDBACK_BUFFER - для ‘сохран’ данных из GS или VS
  - GL_UNIFORM_BUFFER - буфер констант (UBO)
  - http://www.opengl.org/wiki/Buffer_Object

OpenGL buffer object

- Работа с буфером, понятие текущего объекта
  - копирование данных с cpu на gpu

```
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, N*sizeof(GLfloat), (GLfloat*)cpuData, GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```



  - копирование из текстуры ‘tex’ в буфер ‘vbo’

```
glBindBuffer(GL_PIXEL_PACK_BUFFER, vbo);
glBindTexture(GL_TEXTURE_2D, tex);
glGetTexImage(GL_TEXTURE_2D, 0, GL_RGBA, GL_FLOAT, 0);
glBindBuffer(GL_PIXEL_PACK_BUFFER, 0);
```

  - копирование из одного буфера в другой

```
glBindBuffer(GL_ARRAY_BUFFER, vbo1);
glBindBuffer(GL_COPY_WRITE_BUFFER, vbo2);
glCopyBufferSubData(GL_ARRAY_BUFFER, GL_COPY_WRITE_BUFFER,0,0, sizeof(float)*N);
```

// glCopyBufferSubData is available only if the GL version is 3.1 or greater.

OpenGL buffer object

- Цитата из вики

GL_COPY_READ_BUFFER and GL_COPY_WRITE_BUFFER.

These have no particular semantics.

Because they have no actual meaning, they are useful targets for copying buffer object data with glCopyBufferSubData(…).

You do not have to use these targets when copying, but by using them, you avoid disturbing buffer targets that have actual semantics.

```
glBindBuffer(GL_ARRAY_BUFFER, vbo1);
glBindBuffer(GL_COPY_WRITE_BUFFER, vbo2);
glCopyBufferSubData(GL_ARRAY_BUFFER, GL_COPY_WRITE_BUFFER,0,0, sizeof(float)*N);
```

EXT_direct_state_access

- Традиционный метод работы с объектом:

```
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, N*sizeof(GLfloat), (GLfloat*)cpuData, GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

- EXT_direct_state_access

```
glNamedBufferDataEXT (vbo, N*sizeof(GLfloat), (GLfloat*)cpuData, GL_STATIC_DRAW);
```

- В EXT_direct_state_access есть ф-ии для:
  - Работы с буферами
  - Работы с VAO
  - Задания шейдерных переменными (uniform переменных)
  - Работы с FBO (Frame Buffer Object)

- http://steps3d.narod.ru/tutorials/dsa-tutorial.html
- http://www.g-truc.net/post-0279.html

OpenGL, Константы

- GL, в шейдере:
  - Uniform переменные
  - UBO

```
// vertex shader
#version 330

uniform mat4 projectionMatrix;
uniform mat4 modelViewMatrix;

in  vec2 vertex;

void main(void)
{
  vec4 viewPos = modelViewMatrix*vec4(vertex,0.0,1.0);
  gl_Position     = projectionMatrix*viewPos;
}
```

На CPU:

```
float data[3] = {0.5f, 1.0f, 0.1f};
GLint location = glGetUniformLocation(program, “lightDirection”);

// Далее так:
glUseProgram(program);
if(location >= 0)
  glUniform3fv(location, 1, data);

// Или так:
if(location >= 0)
  glProgramUniform3fEXT(program, location, 1, data); // EXT_direct_state_access
```

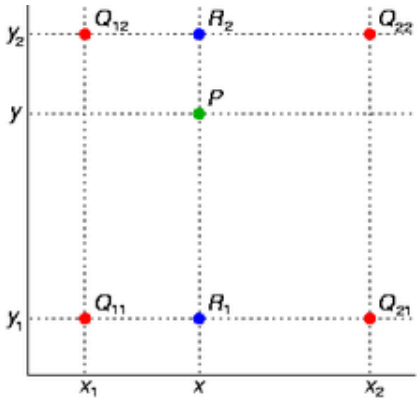
OpenGL, текстуры

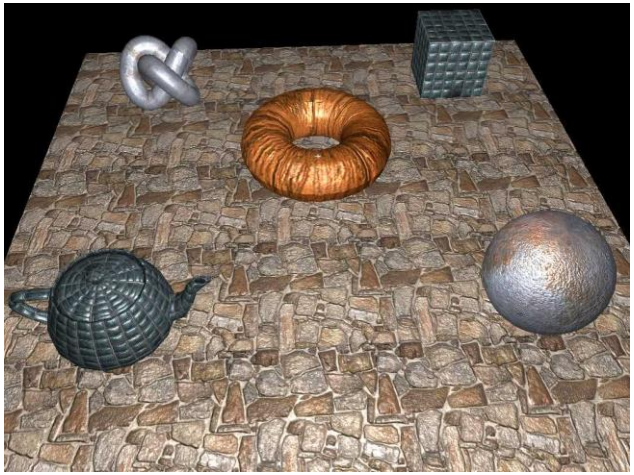
- Текстуры бывают:
  - GL_TEXTURE_1D
  - GL_TEXTURE_2D
  - GL_TEXTURE_3D
  - GL_TEXTURE_BUFFER
  - GL_TEXTURE_CUBE_MAP
  - GL_TEXTURE_1D_ARRAY
  - GL_TEXTURE_2D_ARRAY
  - GL_TEXTURE_CUBE_MAP_ARRAY
  - GL_TEXTURE_2D_MULTISAMPLE
  - GL_TEXTURE_2D_MULTISAMPLE_ARRAY

- Поведение текстуры и буфера отличается тем что:
  - When you use a freshly generated texture name, the first bind helps define the type of the texture.
  - It is not legal to bind an object to a different target than the one it was previously bound with.
- http://www.opengl.org/wiki/Texture

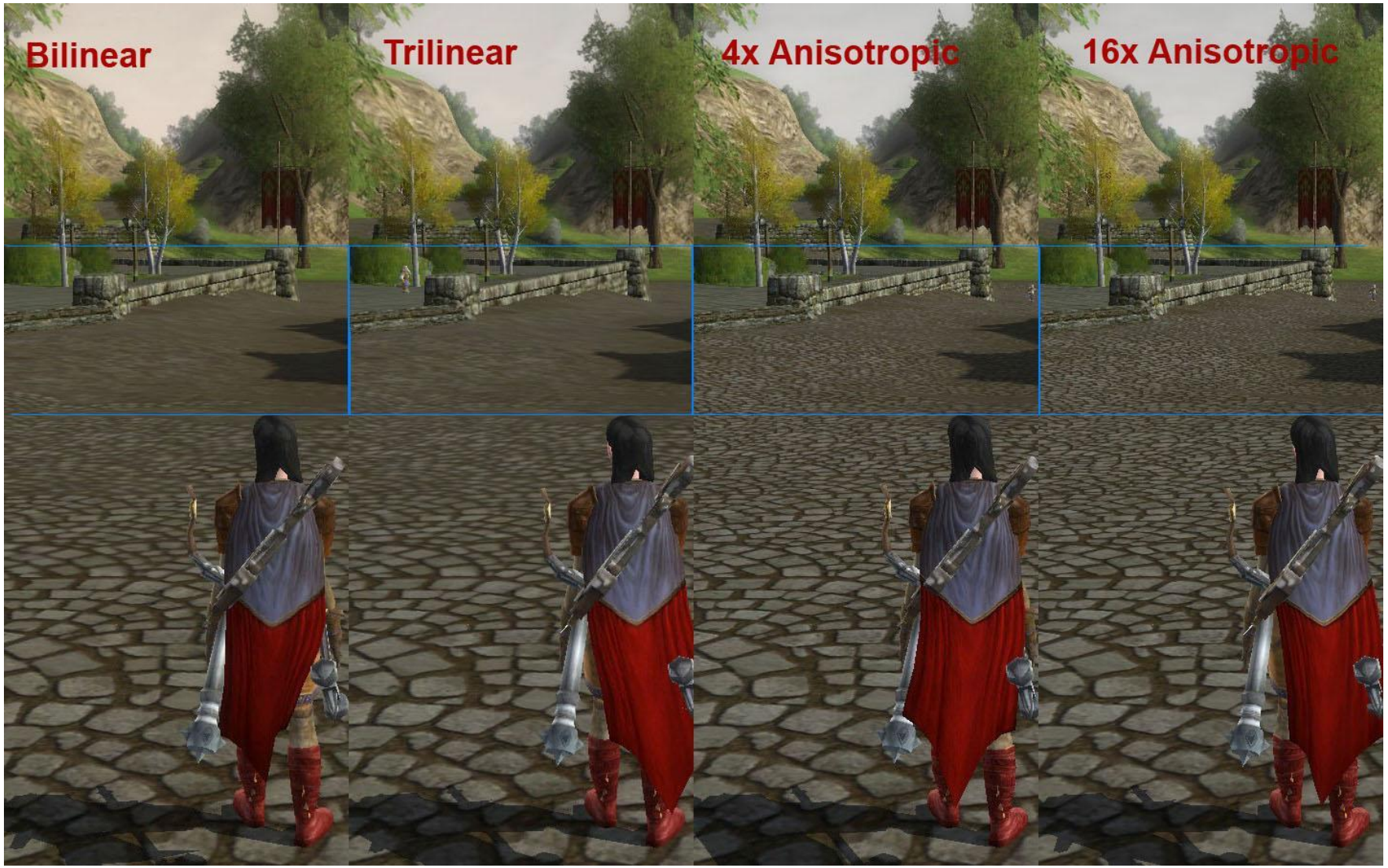
OpenGL, текстуры

- Текстура – хранилище данных
  - Хранилище значений некоторой функции
  - Выборка из текстуры – процесс получения значения это функции
    - Включает в себя интерполяцию, фильтрацию, декомпрессия лету (DXT сжатие)
  - Функция может быть задана на :
    - 1D сетке
    - 2D сетке
    - 3D сетке
    - Кубмап
    - …
  - Фильтрация



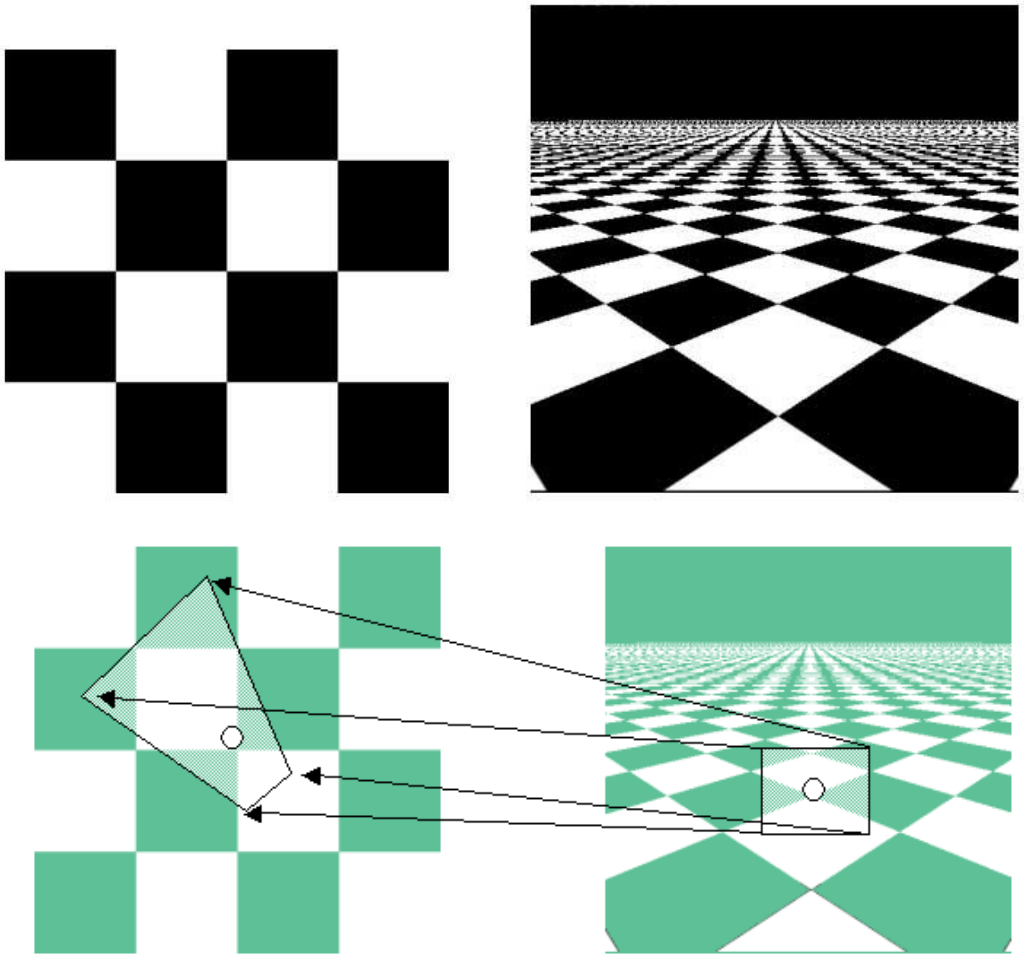


OpenGL, фильтрация текстур

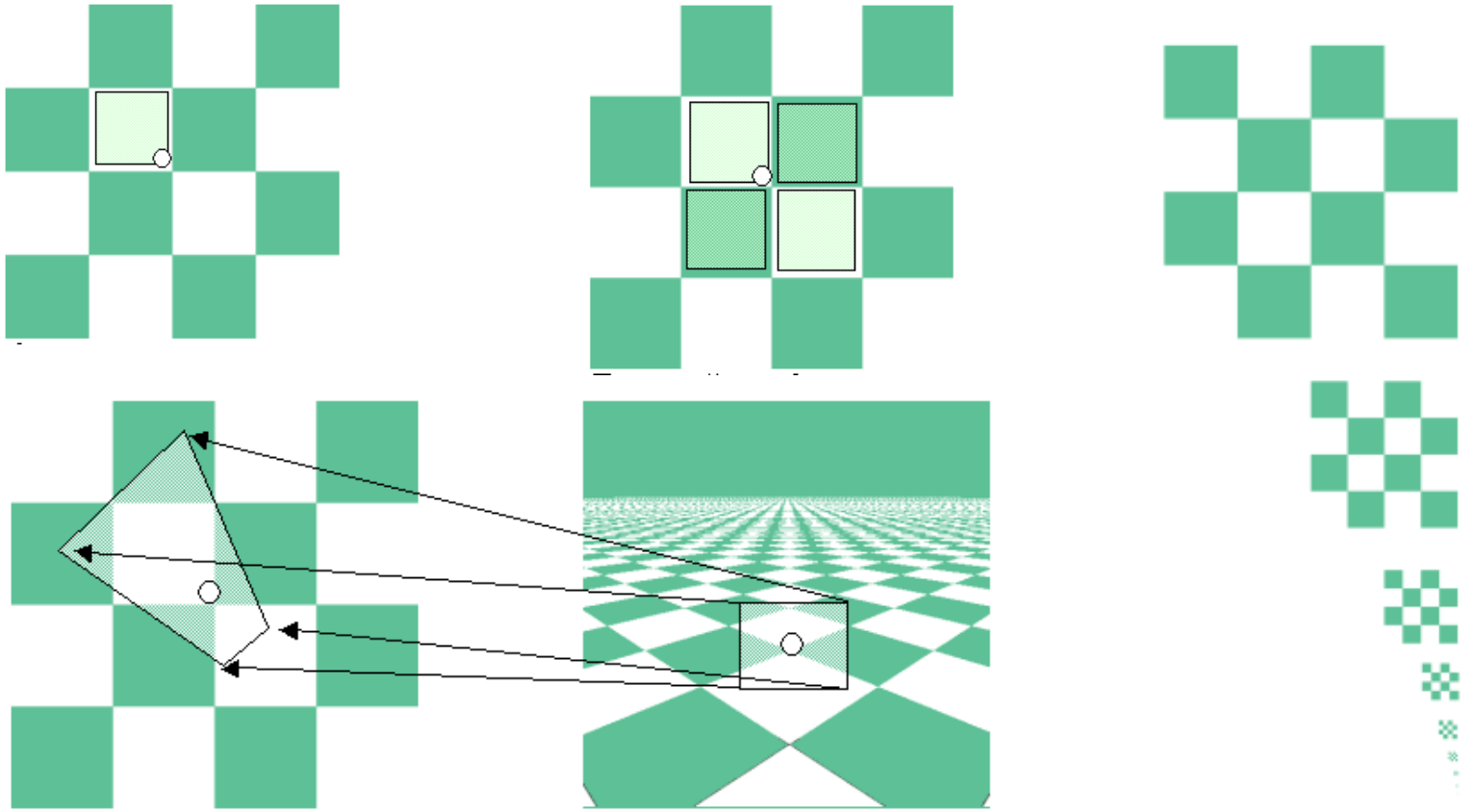


OpenGL, фильтрация текстур

- Почему нужна фильтрация?



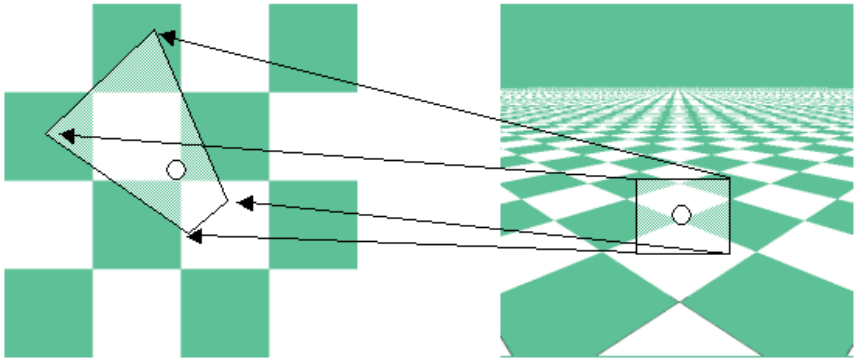
OpenGL, фильтрация текстур



- http://cgm.computergraphics.ru/content/view/56

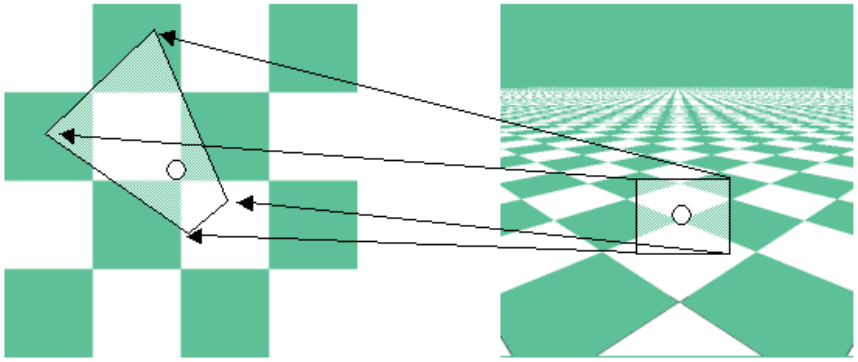
OpenGL, фильтрация текстур

- Как определяют номер мип уровня при фильтрации?



OpenGL, фильтрация текстур

- Как определяют номер мип уровня при фильтрации?



```
void main(void)
{
  vec4 color1 = textureLod(someSampler, textureCoords, 0);
  vec4 color2 = texture     (someSampler, textureCoords);

  vec2 texDx = dFdx(textureCoords);
  vec2 texDy = dFdy(textureCoords);

  vec4 color3 = textureGrad(someSampler, textureCoords, texDx, texDy);
}
```

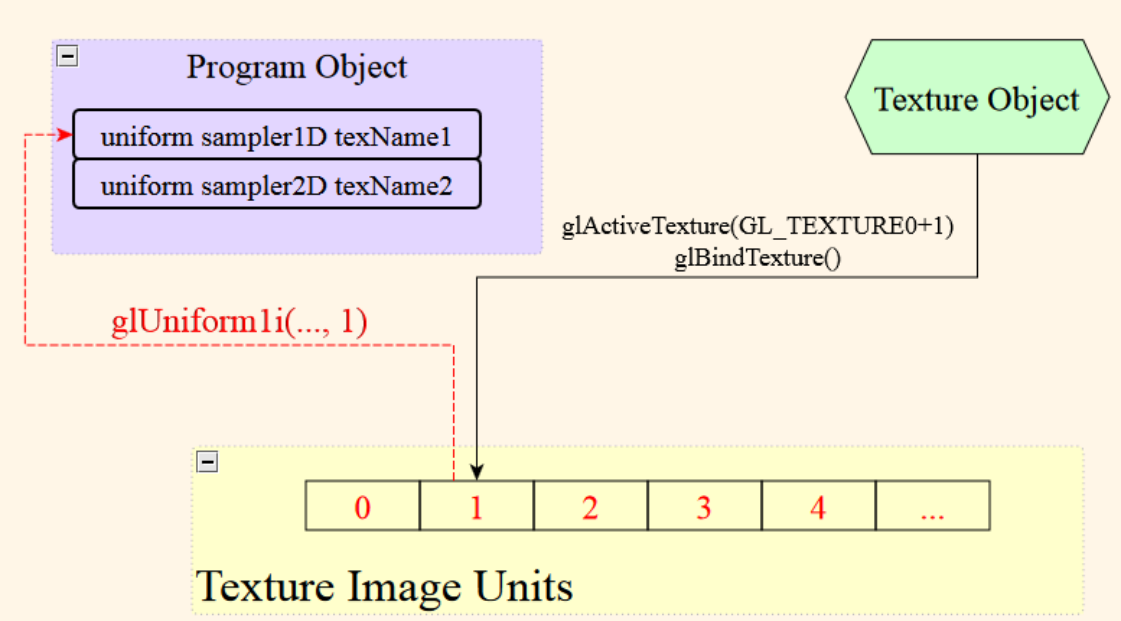
OpenGL, текстуры

- Настройка фильтрации

```
glusLoadTgaImage(file_name.c_str(), &image);

glGenTextures(1, &m_colorTexture);
glBindTexture(GL_TEXTURE_2D, m_colorTexture);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

glTexImage2D(GL_TEXTURE_2D, 0, image.format, image.width, image.height,
                                        0, image.format, GL_UNSIGNED_BYTE, image.data);
glGenerateMipmap(GL_TEXTURE_2D);

glusDestroyTgaImage(&image);
```

Привязка текстур в GL

```
void bindTexture (GLuint program, GLint unit,  const GLchar *name, GLuint texture)
{
  glActiveTexture (GL_TEXTURE0 + unit);
  glBindTexture   (GL_TEXTURE_2D, texture);

  GLint location = glGetUniformLocation (program, name);

  if(location >=0)
    glUniform1i(location, unit);
}
```



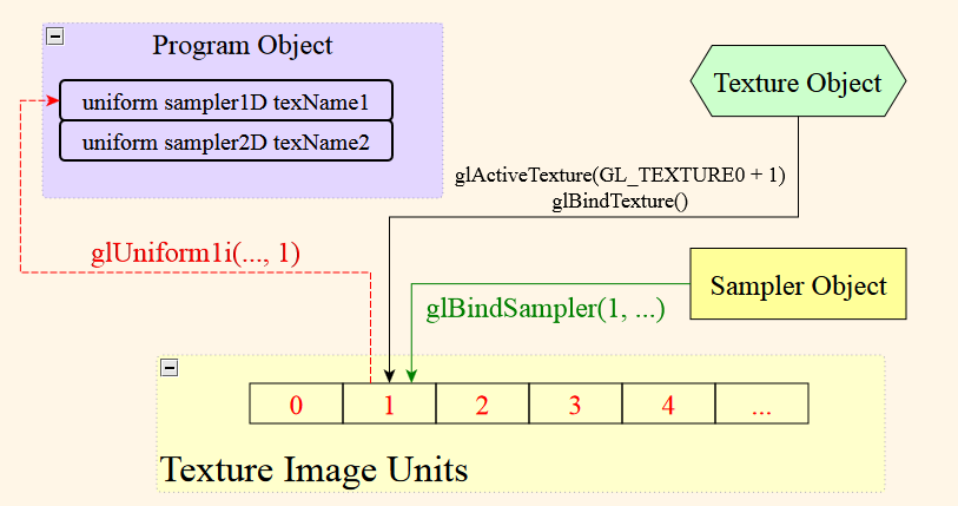
OGL3 glTexParameteri

В OGL3 текстуры и сэмплеры не разделены

```
Texture2D::Texture2D(GLenum format, GLsizei width, GLsizei height, const void* data)
{
  glGenTextures(1, &m_colorTexture);
  glBindTexture(GL_TEXTURE_2D, m_colorTexture);

  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

  glTexImage2D(GL_TEXTURE_2D, 0, format, width, height, 0, format, GL_UNSIGNED_BYTE, data);
  glGenerateMipmap(GL_TEXTURE_2D);
}
```

OpenGL4 Sampler

```
void bindTextureSampler(GLuint program, GLint unit, const GLchar *name,
                                GLuint texture, GLint sampler)
{
  glActiveTexture(GL_TEXTURE0 + unit);
  glBindSampler(unit, sampler);

  glBindTexture(GL_TEXTURE_2D, texture);

  GLint location = glGetUniformLocation(program, name);
  if(location >=0)
    glUniform1i(location, unit);
}
```



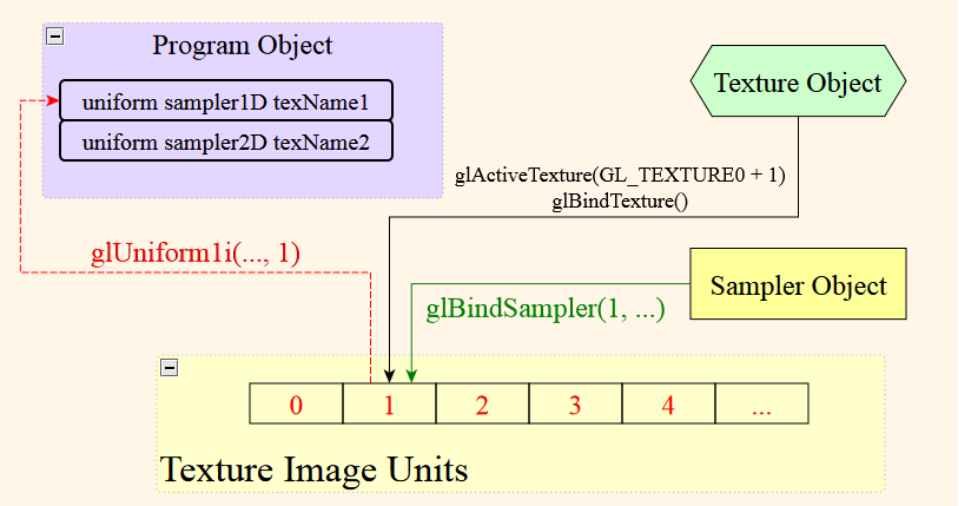
OGL4 Sampler Object

- В OGL3 текстуры и сэмплеры не разделены
- В OGL4 добавлен объект “Sampler”

```
glGenSamplers (1, &s_id);
glSamplerParameteri (s_id, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glSamplerParameteri (s_id, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glSamplerParameteri (s_id, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glSamplerParameteri (s_id, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
…..
…..
glActiveTexture (GL_TEXTURE0 + unit);
glBindSampler (unit, s_id);
```

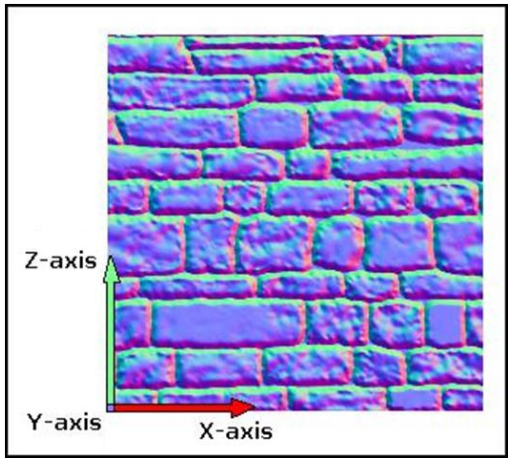
OpenGL4 Sampler

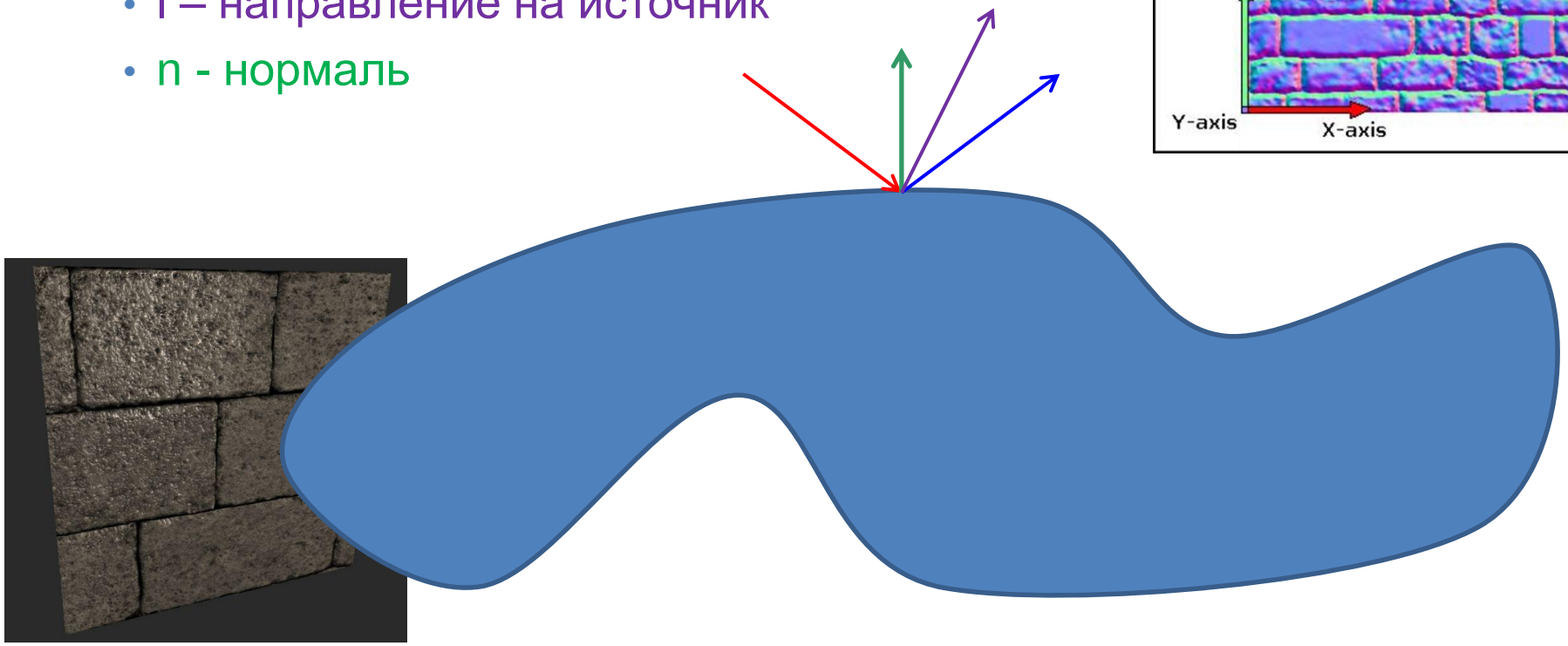
```
void bindTextureSampler(GLuint program, GLint unit, const GLchar *name,
                        GLuint texture, GLint sampler)
{
  glActiveTexture(GL_TEXTURE0 + unit);
  glBindSampler(unit, sampler);

  glBindTexture(GL_TEXTURE_2D, texture);

  GLint location = glGetUniformLocation(program, name);
  if(location >=0)
    glUniform1i(location, unit);
}
```



Bump mappimg (Normal mapping)

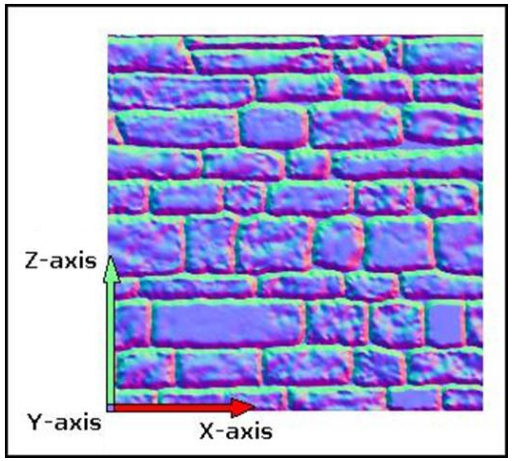
- Идея
  - v – view вектор
  - r – отраженный v вектор
  - l – направление на источник
  - n - нормаль

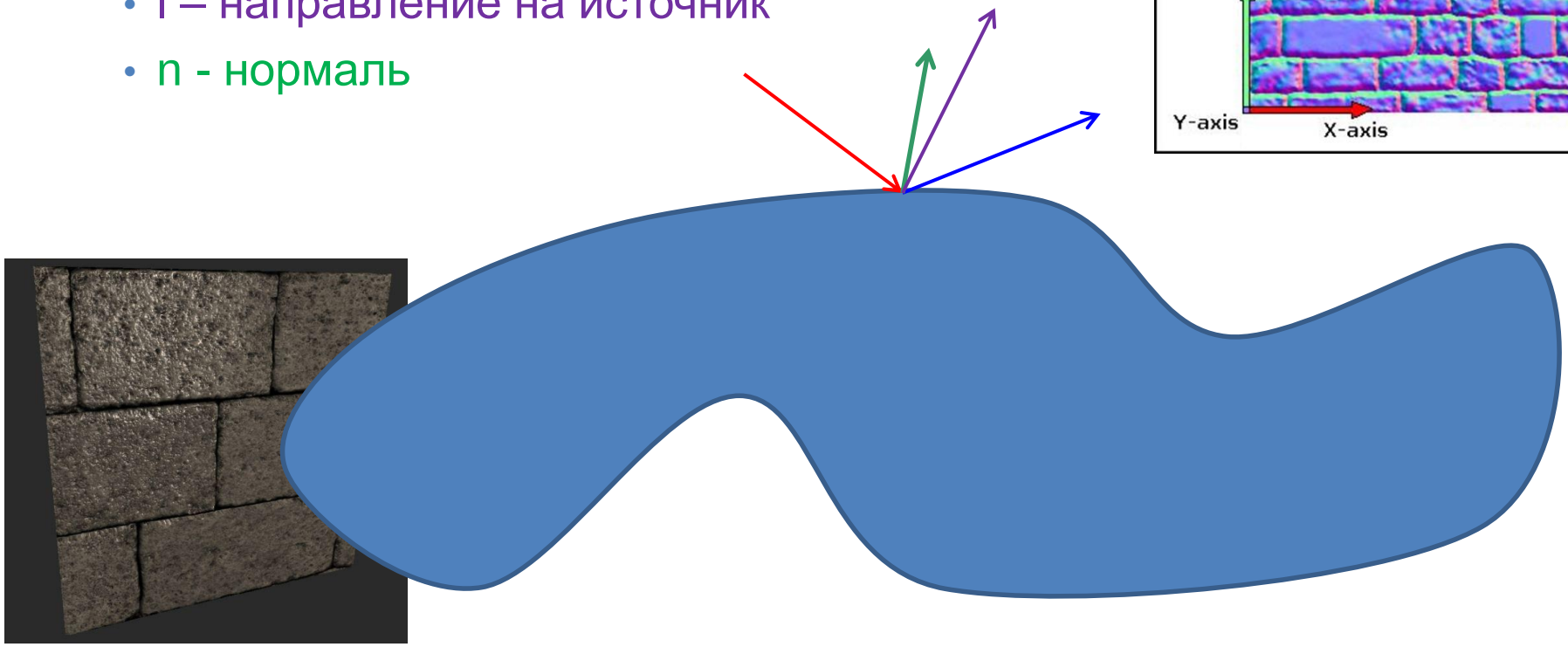




Bump mappimg (Normal mapping)

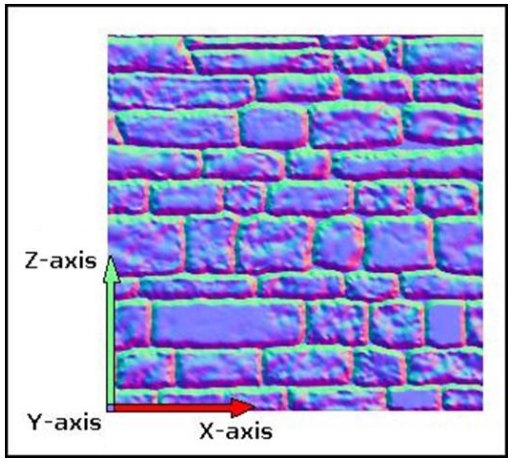
- Идея
  - v – view вектор
  - r – отраженный v вектор
  - l – направление на источник
  - n - нормаль

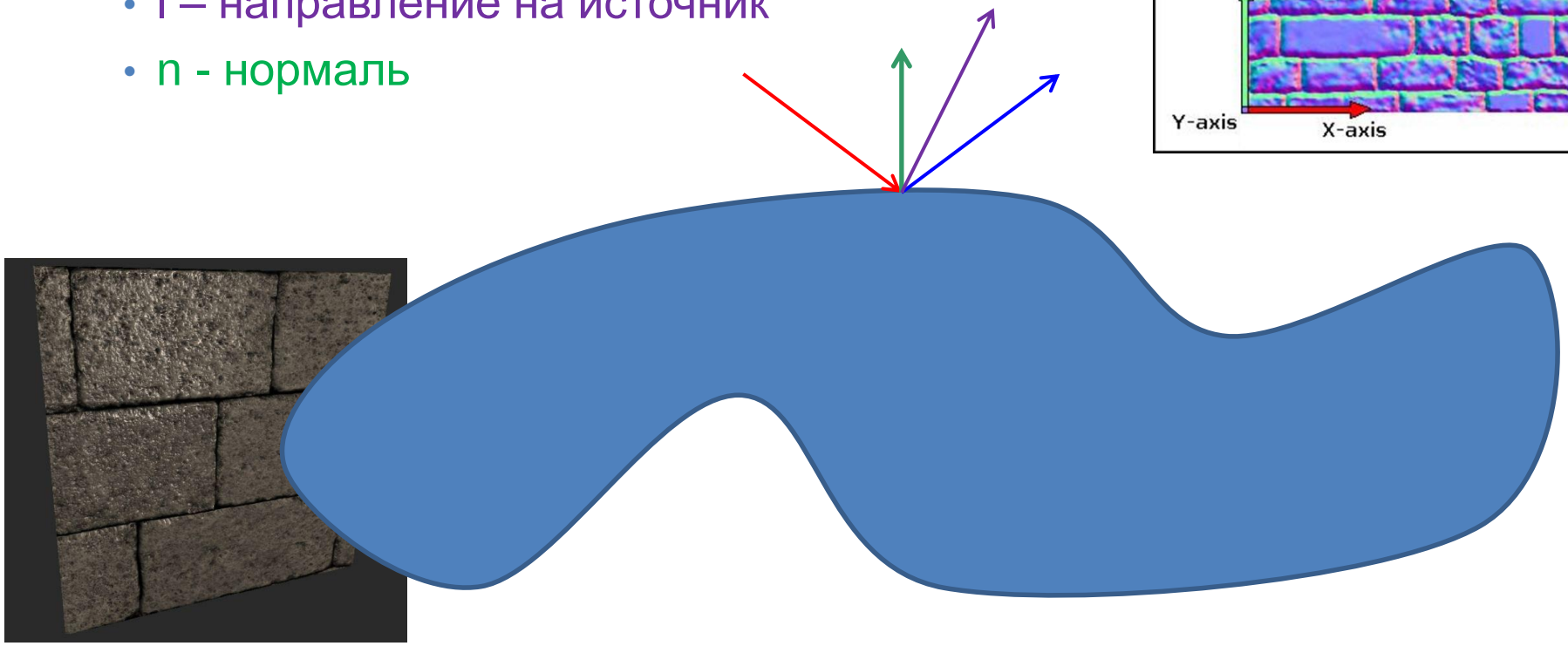




Bump mappimg (Normal mapping)

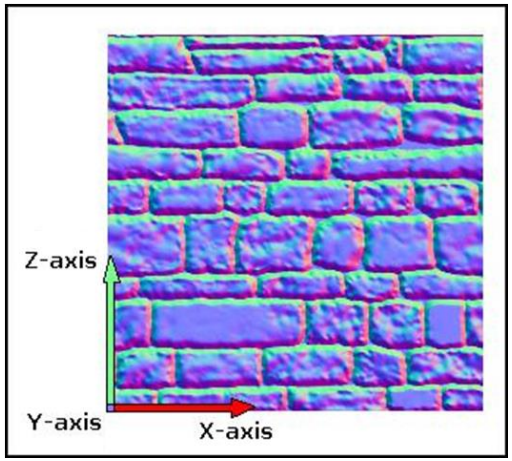
- Идея
  - v – view вектор
  - r – отраженный v вектор
  - l – направление на источник
  - n - нормаль

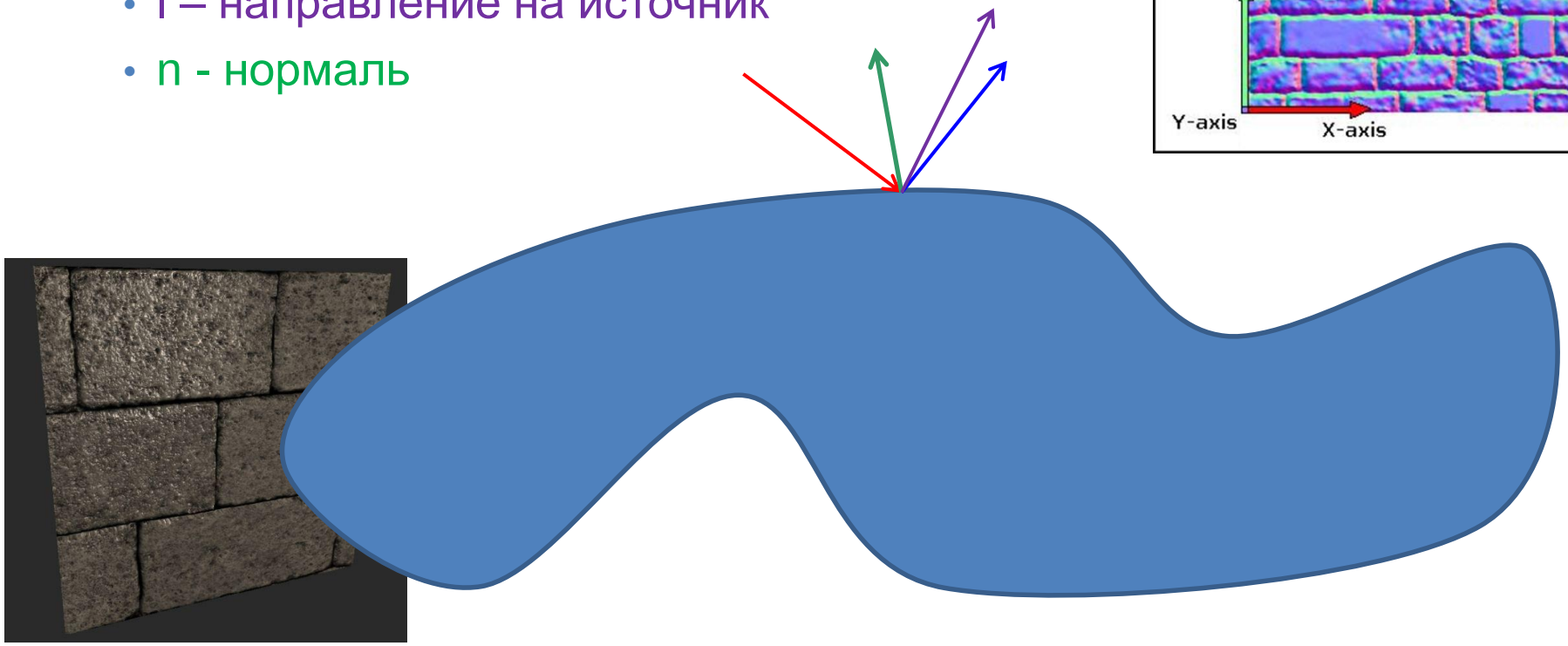




Bump mappimg (Normal mapping)

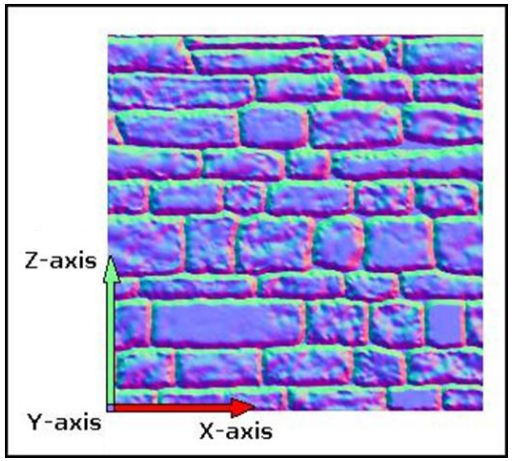
- Идея
  - v – view вектор
  - r – отраженный v вектор
  - l – направление на источник
  - n - нормаль

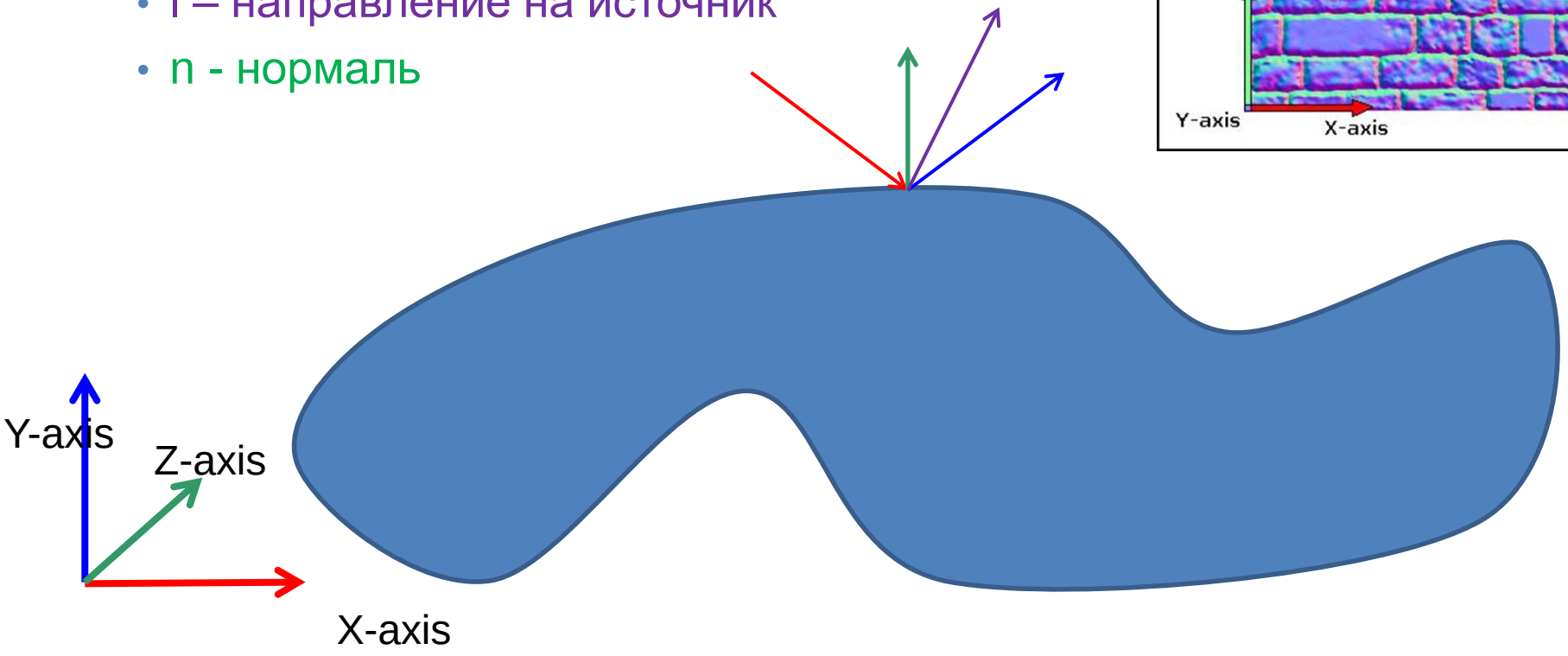




Bump mappimg (Normal mapping)

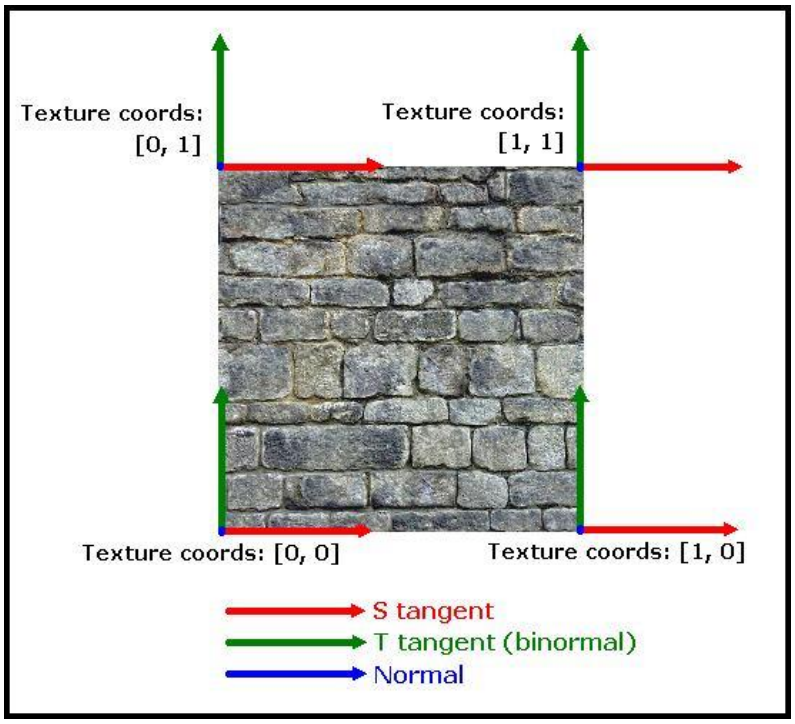
- Идея
  - v – view вектор
  - r – отраженный v вектор
  - l – направление на источник
  - n - нормаль

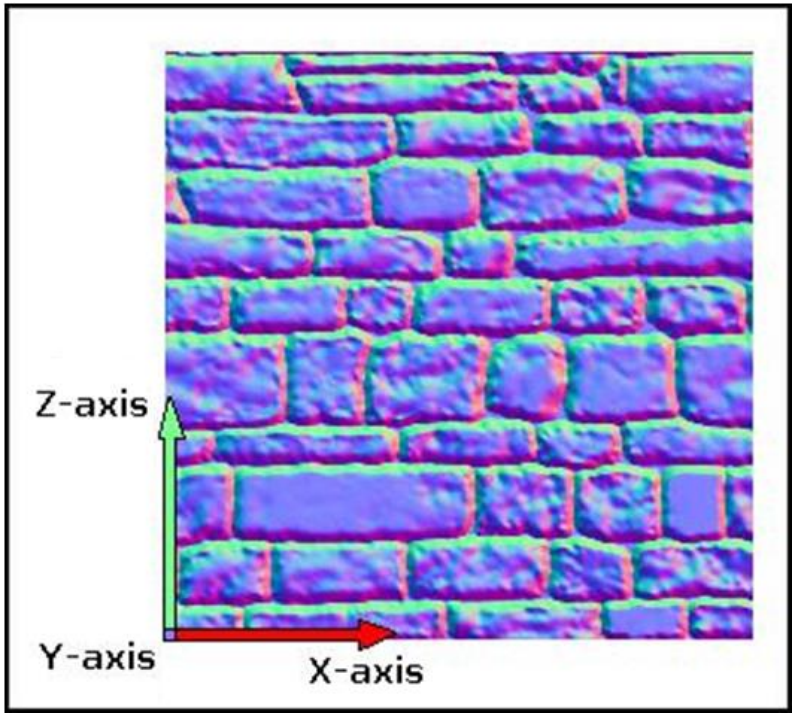




Bump mappimg (Normal mapping)

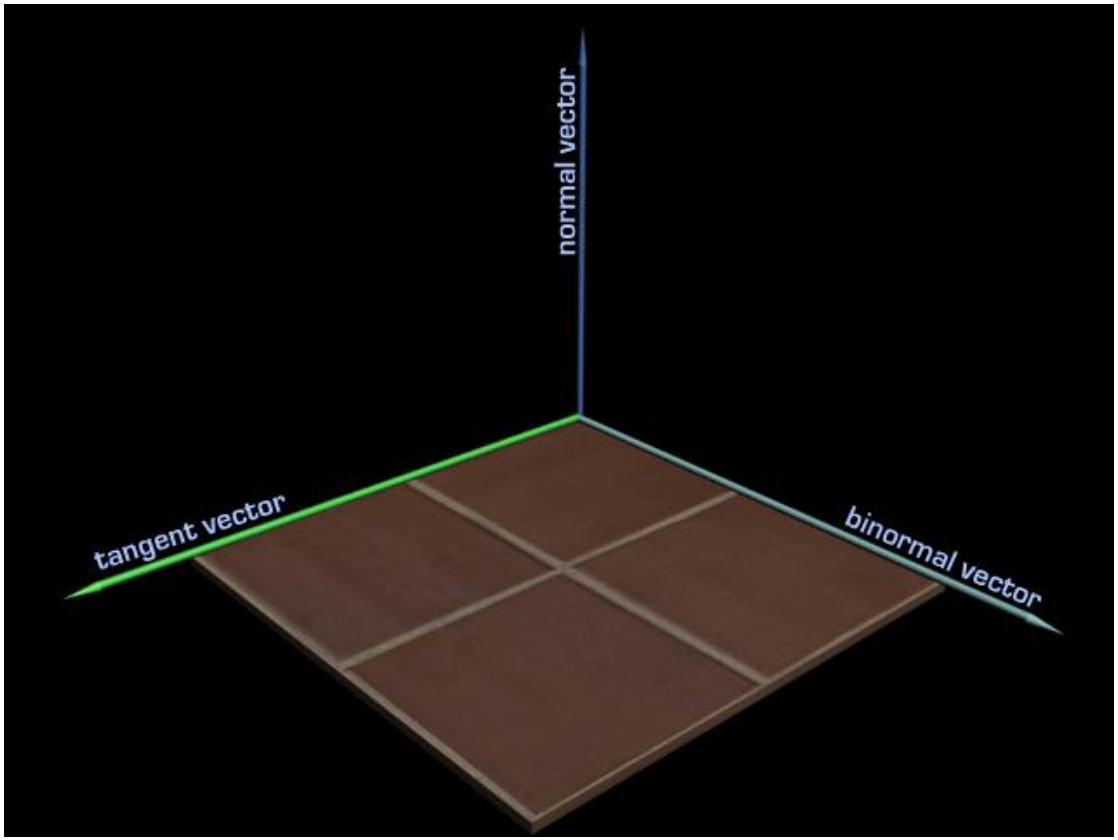
Tangent space

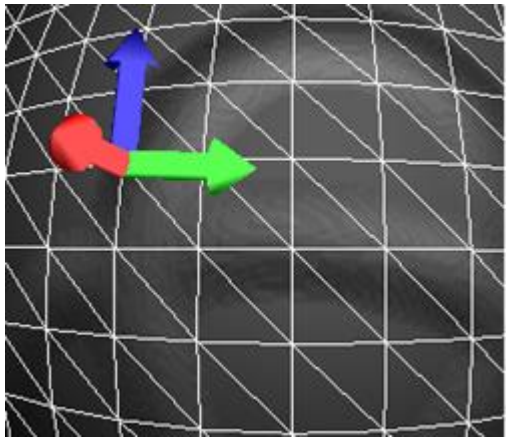




Bump mappimg (Normal mapping)

- Tangent space

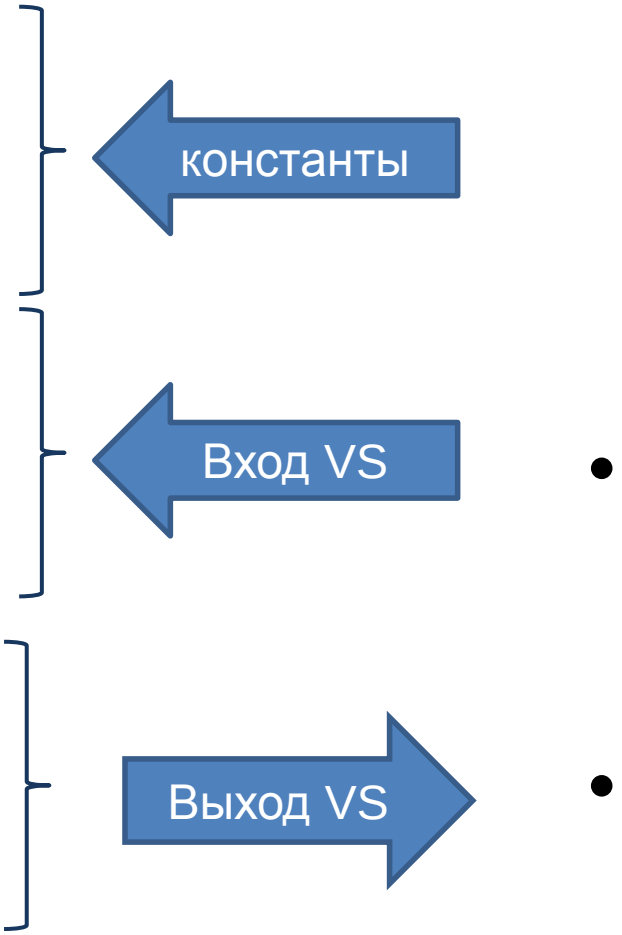




Bump mappimg

- Вершинный шейдер

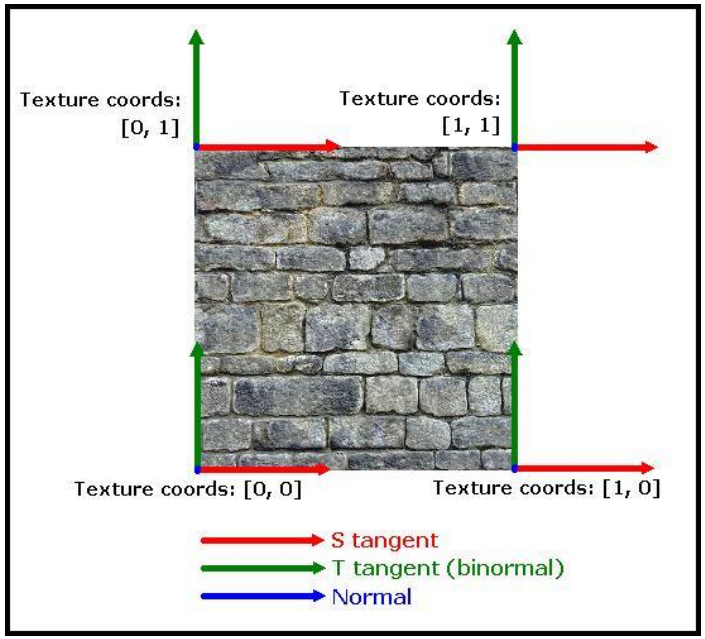
```
uniform mat4 u_projectionMatrix;
uniform mat4 u_modelViewMatrix;
uniform mat3 u_normalMatrix;

uniform vec3 u_lightDirection;

in vec4 a_vertex;
in vec3 a_normal;
in vec3 a_tangent;
in vec3 a_bitangent;
in vec2 a_texCoord;

out vec2 v_texCoord;
out vec3 v_tsLight;
out vec3 v_tsEye;
```



http://nopper.tv/opengl.html

Bump mapping

- Вершинный шейдер

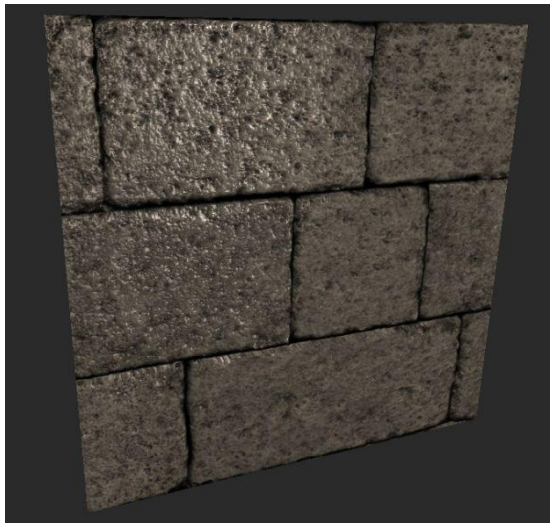



```
void main(void)
{
    v_texCoord = a_texCoord;

    // Bring tangent, bitangent and normal to camera space.
    vec3 tangent   = u_normalMatrix*a_tangent;
    vec3 bitangent = u_normalMatrix*a_bitangent;
    vec3 normal    = u_normalMatrix*a_normal;

    // Brings the light from camera to texture space.
    v_tsLight.x = dot(tangent,   u_lightDirection);
    v_tsLight.y = dot(bitangent, u_lightDirection);
    v_tsLight.z = dot(normal,    u_lightDirection);


    vec4 vertex = u_modelViewMatrix*a_vertex; // Bring vertex in camera space

    vec3 eye = -vec3(vertex); // Calculate eye vector, which is in camera space.

    // Like the light, this brings the eye vector to texture space.
    v_tsEye.x = dot(eye, tangent);
    v_tsEye.y = dot(eye, bitangent);
    v_tsEye.z = dot(eye, normal);

    gl_Position = u_projectionMatrix*vertex;
}
```

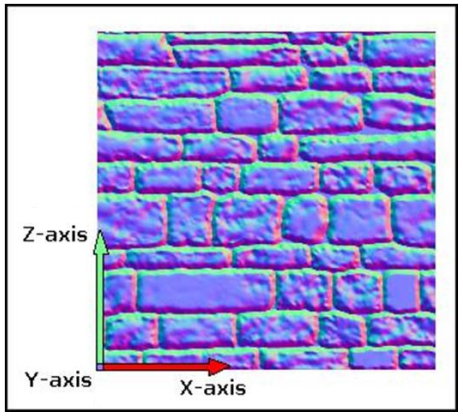
http://nopper.tv/opengl.html

Bump mapping

- Фрагментный шейдер



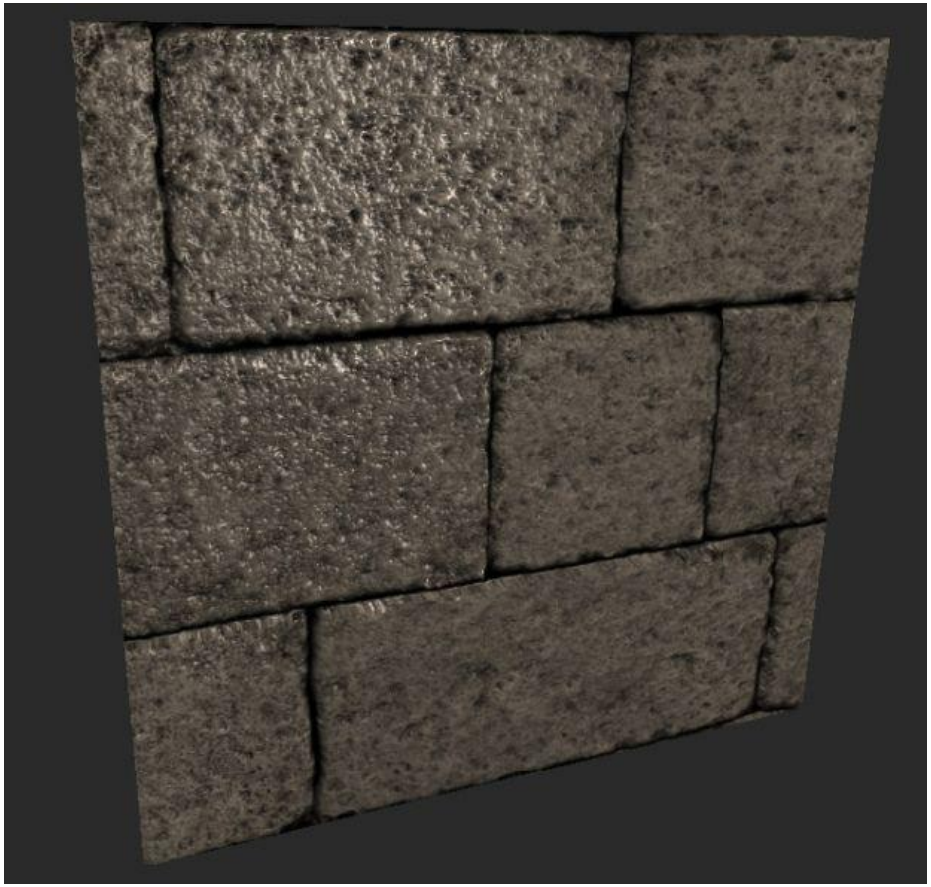
```
uniform sampler2D u_texture;
uniform sampler2D u_normalMap;

in vec2 v_texCoord;
in vec3 v_tsLight;
in vec3 v_tsEye;

out vec4 fragColor;

void main(void)
{
    vec4 color = 0.3 * texture(u_texture, v_texCoord); // Fixed 0.3 ambient light.

    vec3 normal = (texture(u_normalMap, v_texCoord).xyz * 2.0 - 1.0 )*vec3(1.0f, -1.0f, 1.0f);

    vec3 light   = normalize(v_tsLight);

    float nDotL = max(dot(light, normal), 0.0);

    if (nDotL > 0.0)
    {
        color += ...; // расчет освещения по фонгу
    }

    fragColor = color;
}
```

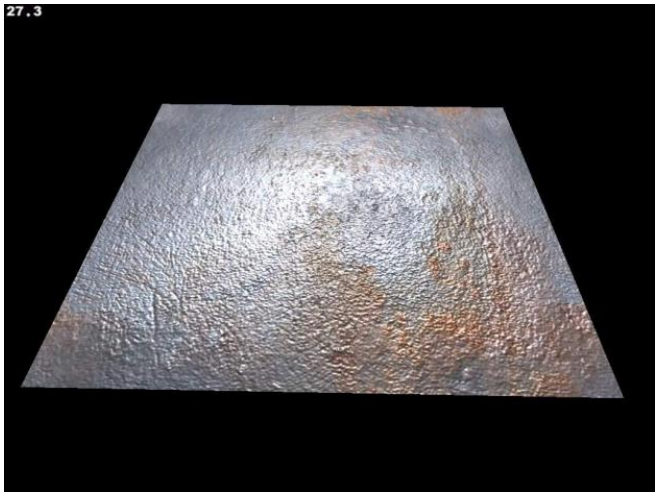


http://nopper.tv/opengl.html

Bump mappimg (Normal mapping)

- http://nopper.tv/opengl.html (lesson 7)

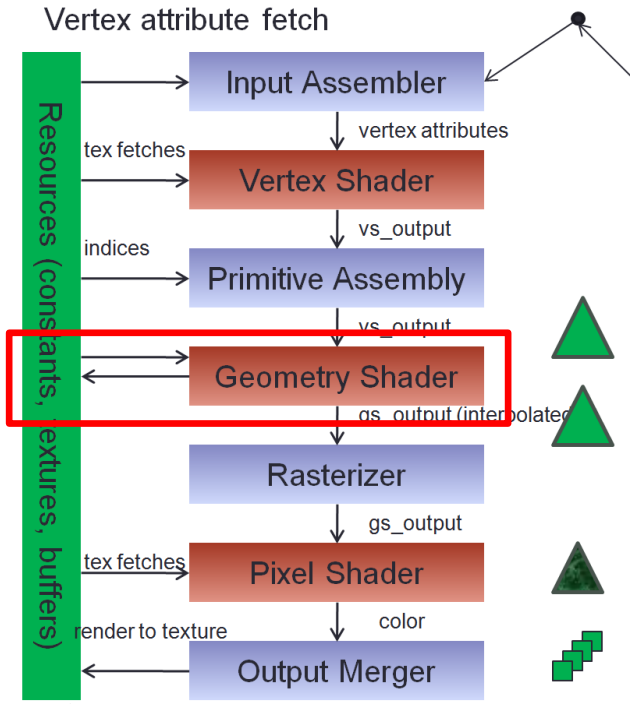




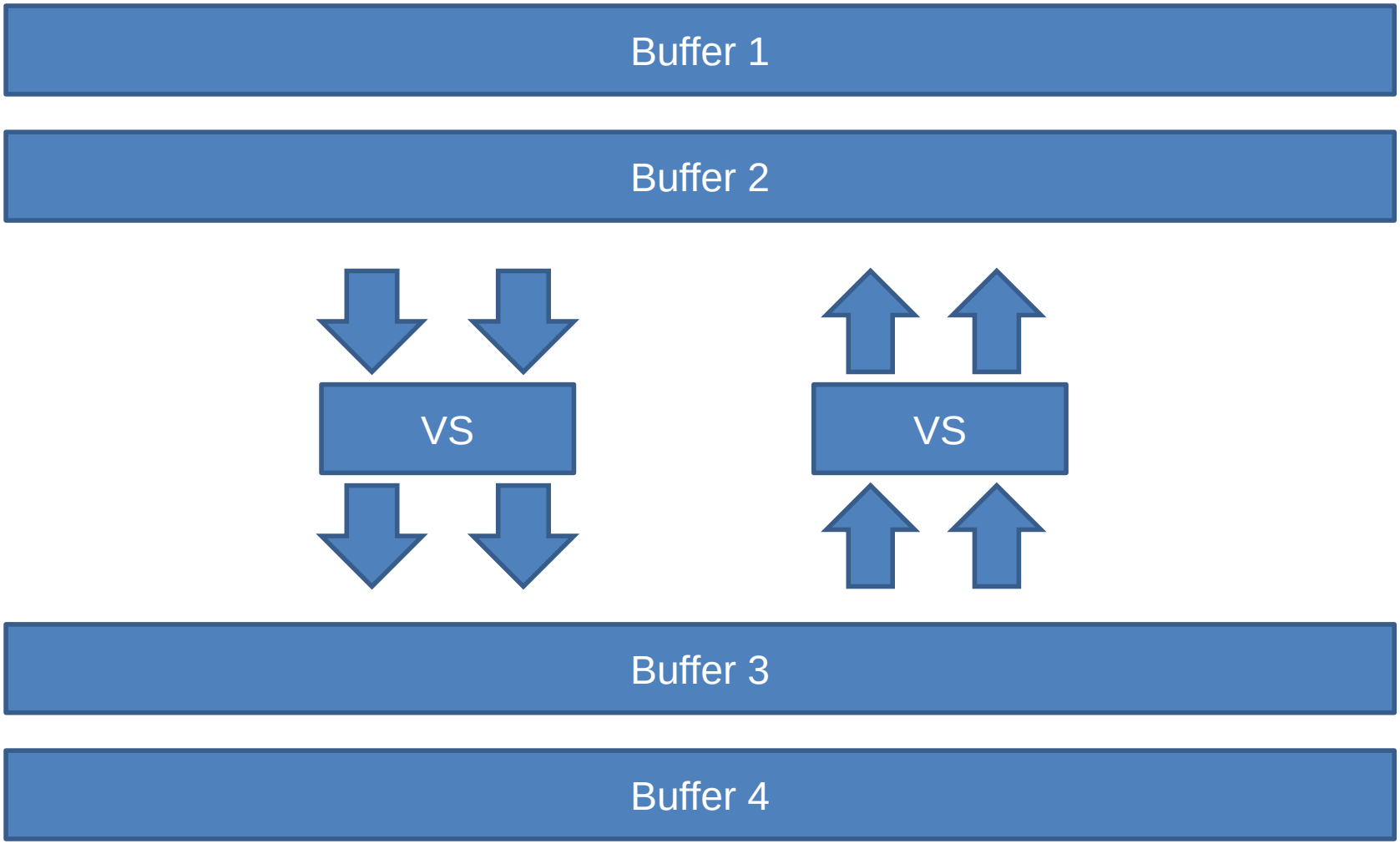
Transform Feedback, append-buffer

- DX: Stream Out
- GL: Transform Feedback
- Результат работы геометрического шейдера может быть сохранен в DRAM память.

- Геометрического шейдера может не быть
  - (и пиксельного тоже)
  - Можно отключить растеризатор
  - Получится что-то похожее на CUDA kernel

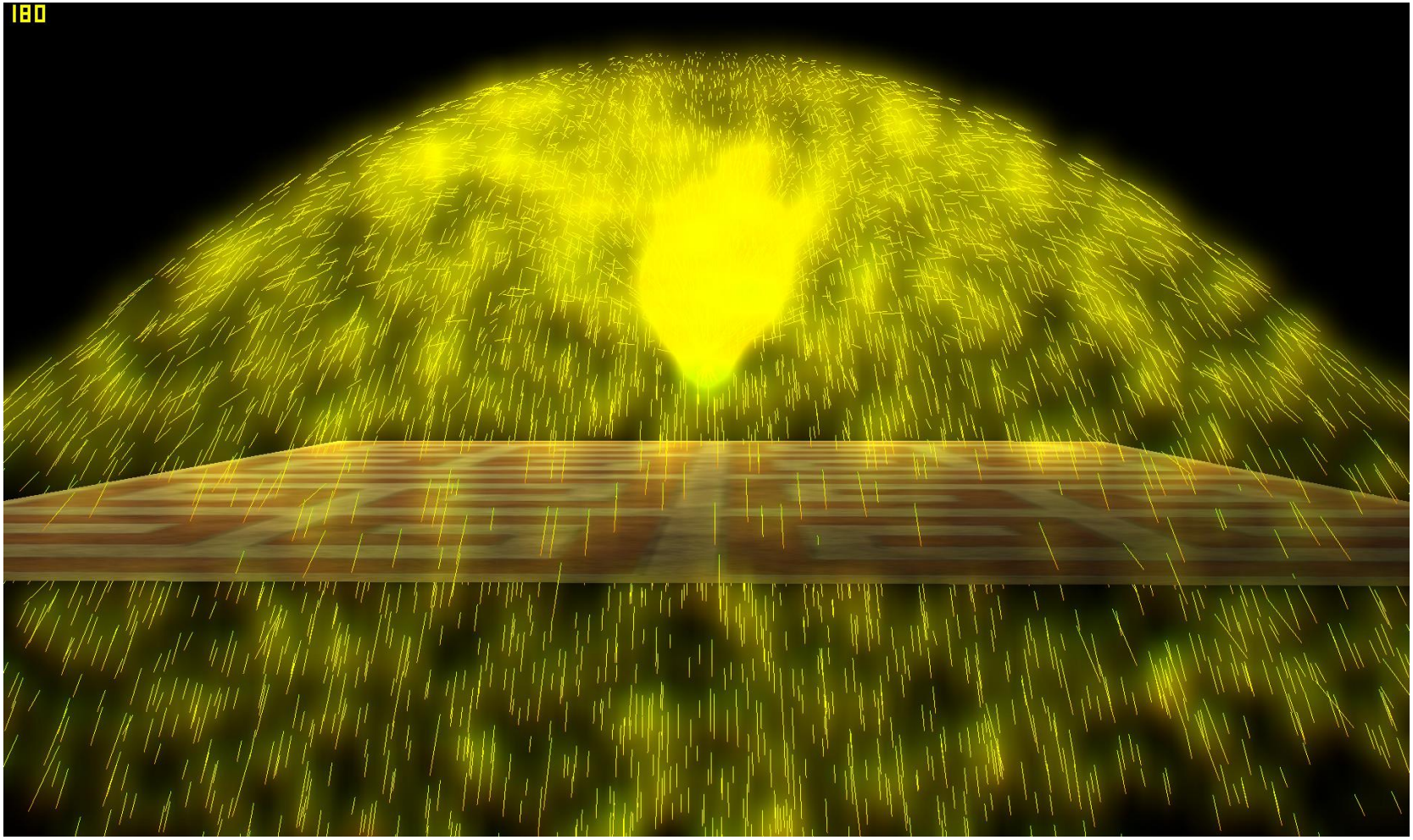


Stream Out & Transform feedback



Stream Out & Transform feedback

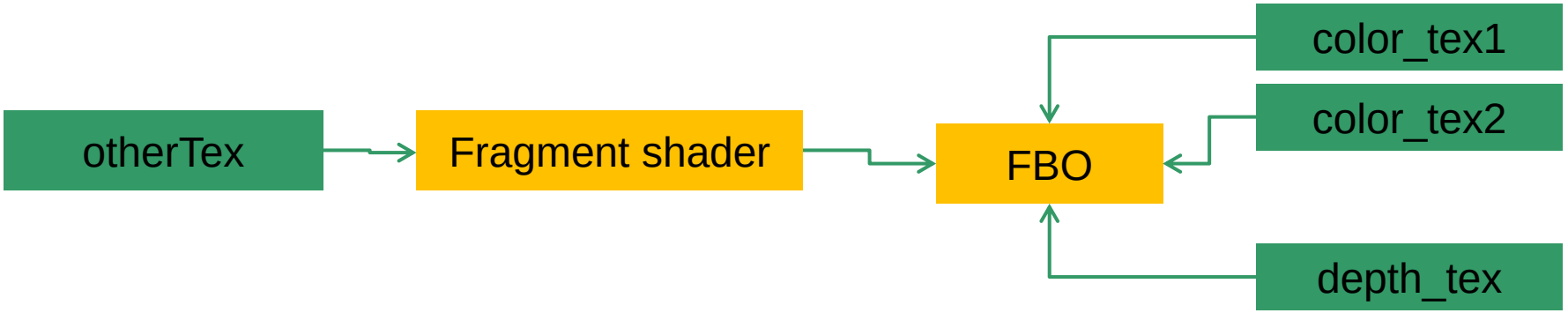
- Рассчет физики частиц



Frame Buffer Object

- Рендеринг изображения в текстуру
- FBO – легковесный объект к которому приаттачены:
  - N текстур цвета
  - 1 текстура с буфером глубины

```
glBindFramebuffer(GL_FRAMEBUFFER, fbo);
```

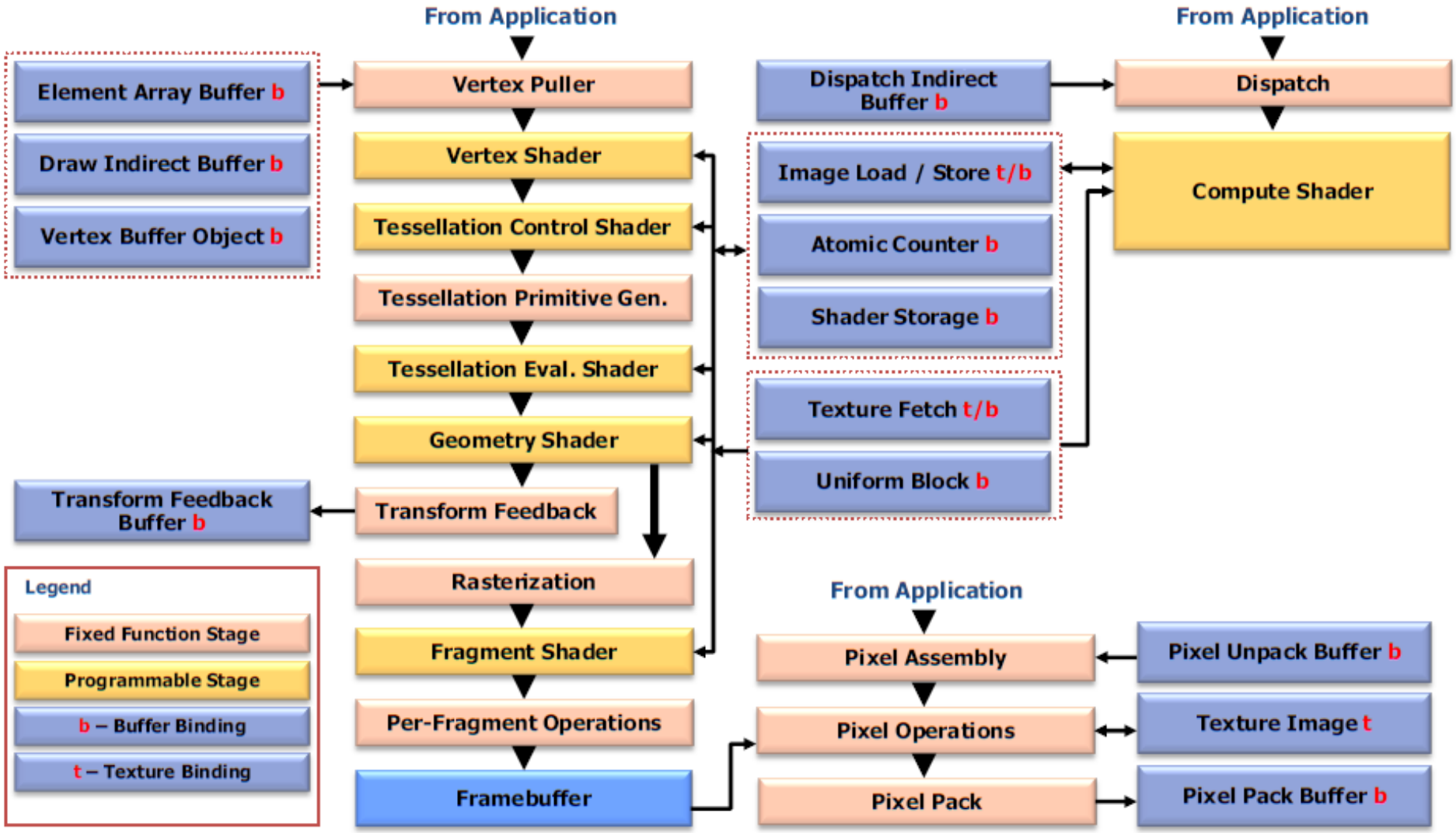


- http://www.opengl.org/wiki/Framebuffer_Object

VS

OpenGL Compute

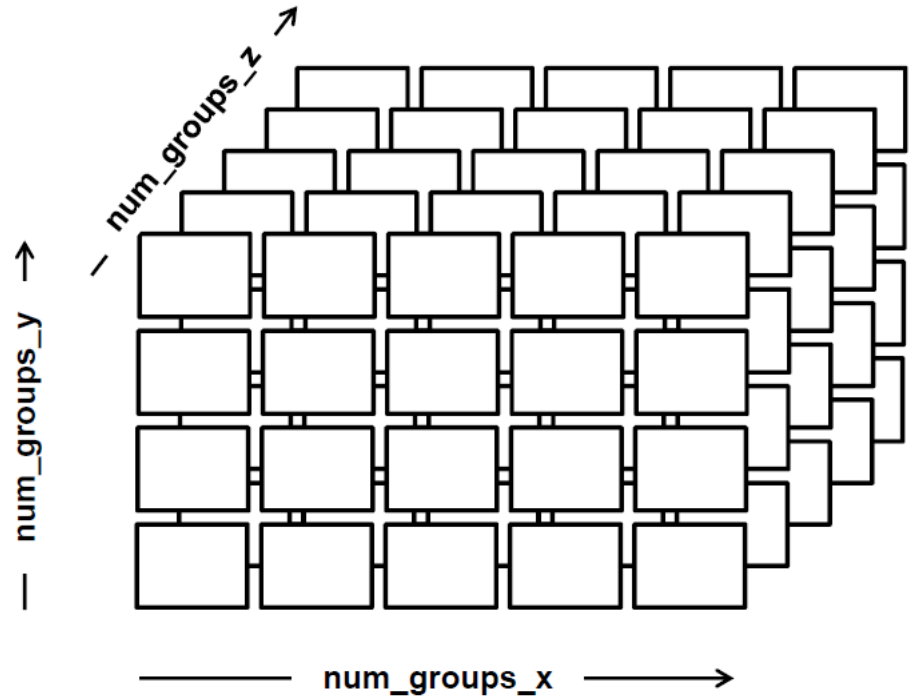
OpenGL 4.3


From Application
Element Array Buffer b
Draw Indirect Buffer b
Vertex Buffer Object b
Vertex Puller
Vertex Shader
Tessellation Control Shader
Tessellation Primitive Gen.
Tessellation Eval. Shader
Geometry Shader
Transform Feedback
Transform Feedback Buffer b
Rasterization
Fragment Shader
Per-Fragment Operations
Framebuffer
From Application
Dispatch Indirect Buffer b
Dispatch
Compute Shader
Image Load / Store t/b
Atomic Counter b
Shader Storage b
Texture Fetch t/b
Uniform Block b
From Application
Pixel Assembly
Pixel Unpack Buffer b
Pixel Operations
Texture Image t
Pixel Pack
Pixel Pack Buffer b
Legend
Fixed Function Stage
Programmable Stage
b – Buffer Binding
t – Texture Binding

GPGPU

- CUDA, OpenCL, Direct Compute, OpenGL Compute

```
void glDispatchCompute( num_groups_x, num_groups_y, num_groups_z );
```

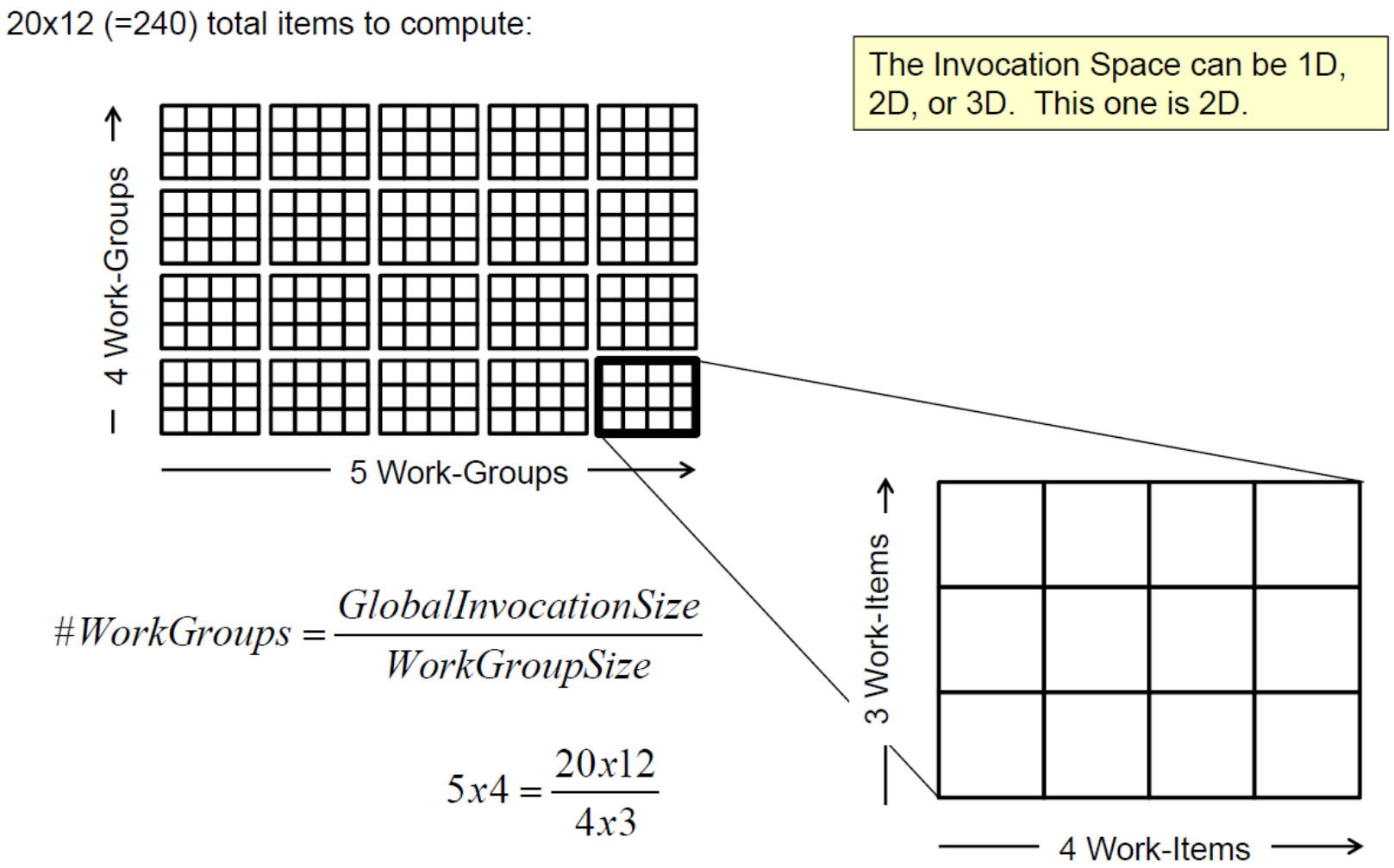


If the problem is 2D, then num_groups_z = 1

If the problem is 1D, then num_groups_y = 1 and num_groups_z = 1

GPGPU

- CUDA, OpenCL, Direct Compute, OpenGL Compute



CUDA VectorAdd (CUDA_SDK)

```
float* h_A = (float*)malloc(size);
float* h_B = (float*)malloc(size);
float* h_C = (float*)malloc(size);

// Initialize input vectors
RandomInit(h_A, N);
RandomInit(h_B, N);

// Allocate vectors in device memory
cudaMalloc((void**)&d_A, size);
cudaMalloc((void**)&d_B, size);
cudaMalloc((void**)&d_C, size);

// Copy vectors from host memory to device memory
cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

 // Invoke kernel
 int threadsPerBlock = 256;
 int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;

 VecAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);

 // Copy result from device memory to host memory
 // h_C contains the result in host memory
 cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
```

```
// Device code
__global__ void VecAdd(const float* A, const float* B, float* C, int N)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}
```

Вычислительные шейдеры (GL)

- Shader

```
#version 430 core

layout(binding = 1)  buffer inputBuffer
{
    float Input[];
} In;

layout(binding = 2) buffer outputBuffer
{
    float Ouput[];
} Out;

layout (local_size_x = 256, local_size_y = 1, local_size_z = 1) in;

void main(void)
{
    Out.Ouput[gl_GlobalInvocationID.x] = 2.0*In.Input[gl_GlobalInvocationID.x];
}
```

Вычислительные шейдеры (GL)

Создание буферов, копирование данных с CPU на GPU

```
int N = 1024;
std::vector<float> array1(N);
std::vector<float> array2(N);

for(int i=0;i<array1.size();i++)
{
  array1[i] = float(i)/10.0f;
  array2[i] = 0.0f;
}
```

```
cudaMalloc((void**)&buf1 array1.size());
cudaMemcpy(buf1, &array1[0], size, cudaMemcpyHostToDevice);
```

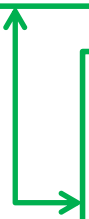
```
GLuint buf1;
glGenBuffers(1, &buf1);
glBindBuffer  (GL_SHADER_STORAGE_BUFFER, buf1);
glBufferData (GL_SHADER_STORAGE_BUFFER, array1.size()*sizeof(float), &array1[0], GL_STATIC_DRAW );
```

```
GLuint buf2;
glGenBuffers(1, &buf2);
glBindBuffer(GL_SHADER_STORAGE_BUFFER, buf2
glBufferData(GL_SHADER_STORAGE_BUFFER, array2.size()*sizeof(float), &array2[0], GL_STATIC_DRAW);
```

Вычислительные шейдеры (GL)

- Запуск вычислительного шейдера

```
glUseProgram(a_prog);

glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 1, buf1);
glBindBufferBase(GL_SHADER_STORAGE_BUFFER, 2, buf2);

glDispatchCompute(array1.size()/256, 1, 1);
glMemoryBarrier(GL_SHADER_STORAGE_BARRIER_BIT);
```



```
int threadsPerBlock = 256;
int blocksPerGrid     = N/ threadsPerBlock;

MyKernel<<<blocksPerGrid, threadsPerBlock>>>(buf1, buf2);
cudaThreadSynchronize();
```

Вычислительные шейдеры (GL)

- Считывание данных на CPU

```
glBindBuffer(GL_SHADER_STORAGE_BUFFER,  buf2);
void* resultData = glMapBufferRange( GL_SHADER_STORAGE_BUFFER,
                                     0, array2.size()*sizeof(float),
                                     GL_MAP_READ_BIT);

memcpy(&array2[0], resultData, array2.size()*sizeof(float));
glUnmapBuffer(GL_SHADER_STORAGE_BUFFER);
```

```
std::cout << "CS, output buffer values: " << std::endl;
  for(int i=0;i<10;i++)
    std::cout << array2[i] << std::endl;

glDeleteBuffers(1, &buf1);
glDeleteBuffers(1, &buf2);
```

```
cudaMemcpy(&array2[0], buf2,
                  array2.size()*sizeof(float),
                  cudaMemcpyDeviceToHost);
```

Будет ли программа работать?

Создание буферов, копирование данных с CPU на GPU

```
int N = 1024;
std::vector<float> array1(N);
std::vector<float> array2(N);

for(int i=0;i<array1.size();i++)
{
  array1[i] = float(i)/10.0f;
  array2[i] = 0.0f;
}

GLuint buf1;
glGenBuffers(1, &buf1);
glBindBuffer(GL_SHADER_STORAGE_BUFFER, buf1);
glBufferData(GL_SHADER_STORAGE_BUFFER, array1.size()*sizeof(float), &array1[0], GL_STATIC_DRAW );

GLuint buf2;
glGenBuffers(1, &buf2);
glBindBuffer(GL_SHADER_STORAGE_BUFFER, buf2) ;
glBufferData(GL_SHADER_STORAGE_BUFFER, array2.size()*sizeof(float), &array2[0], GL_STATIC_DRAW);
```

Будет ли программа работать?

Создание буферов, копирование данных с CPU на GPU

```
int N = 1024;
std::vector<float> array1(N);
std::vector<float> array2(N);

for(int i=0;i<array1.size();i++)
{
  array1[i] = float(i)/10.0f;
  array2[i] = 0.0f;
}

GLuint buf1;
glGenBuffers(1, &buf1);
glBindBuffer(GL_SHADER_STORAGE_BUFFER, buf1);
glBufferData(GL_SHADER_STORAGE_BUFFER, array1.size()*sizeof(float), &array1[0], GL_STATIC_DRAW );

GLuint buf2;
glGenBuffers(1, &buf2);
glBindBuffer(GL_ARRAY_BUFFER, buf2);
glBufferData(GL_ARRAY_BUFFER, array2.size()*sizeof(float), &array2[0], GL_STATIC_DRAW);
```

```
glNamedBufferDataEXT(buf2, array2.size()*sizeof(float), &array2[0], GL_STATIC_DRAW);
```

Отладка в OpenGL

```
#define CHECK_GL_ERRORS ThrowExceptionOnGLError(__LINE__, __FILE__)

static void ThrowExceptionOnGLError (int line, const char *file)
{
  static char errMsg[512];

  GLenum gl_error = glGetError();

  if(gl_error == GL_NO_ERROR)
    return;

  switch(gl_error)
  {
  case GL_INVALID_ENUM:
    sprintf(errMsg, "GL_INVALID_ENUM file %s line %d\n", file, line);
    break;
  …
  case GL_NO_ERROR:
    break;

  default:
    sprintf(errMsg, "Unknown error @ file %s line %d\n",  file, line);
    break;
  }

  if(gl_error != GL_NO_ERROR)
    throw std::runtime_error(errMsg);
}
```

Отладка в OpenGL

- glGetError()
- Удобно использовать макрос типа CHECK_GL_ERROR
- Предположим glTexImage2D сгенерировала ошибку GL_INVALID_VALUE

GL_INVALID_VALUE is generated if ***width*** is less than 0 or greater than **GL_MAX_TEXTURE_SIZE**.

GL_INVALID_VALUE is generated if ***target*** is not **GL_TEXTURE_1D_ARRAY** or **GL_PROXY_TEXTURE_1D_ARRAY** and ***height*** is

GL_INVALID_VALUE is generated if ***target*** is **GL_TEXTURE_1D_ARRAY** or **GL_PROXY_TEXTURE_1D_ARRAY** and ***height*** is les

GL_INVALID_VALUE is generated if ***level*** is less than 0.

GL_INVALID_VALUE may be generated if ***level*** is greater than $log_2(max)$, where *max* is the returned value of **GL_MAX_TEXTURE_**

GL_INVALID_VALUE is generated if ***internalFormat*** is not one of the accepted resolution and format symbolic constants.

GL_INVALID_VALUE is generated if ***width*** or ***height*** is less than 0 or greater than **GL_MAX_TEXTURE_SIZE**.

GL_INVALID_VALUE is generated if non-power-of-two textures are not supported and the ***width*** or ***height*** cannot be represented

GL_INVALID_VALUE is generated if ***border*** is not 0.

OpenGL Debug Output

- Debug Layer, необходимо правильно создать контекст!

```
int attribList[] =
{
  WGL_CONTEXT_MAJOR_VERSION_ARB, 1,
  WGL_CONTEXT_MINOR_VERSION_ARB,  0,
  WGL_CONTEXT_FLAGS_ARB,  WGL_CONTEXT_FORWARD_COMPATIBLE_BIT_ARB |
                                    WGL_CONTEXT_DEBUG_BIT_ARB,
  0, 0, 0
};

attribList[1] = g_major;
attribList[3] = g_minor;

if (!(hRCTemp = wglCreateContextAttribsARB(g_hDC, 0, attribList)))
{
  destroyWindow();
  return false;
}
```

OpenGL Debug Output

- Debug Layer, необходимо правильно создать контекст!

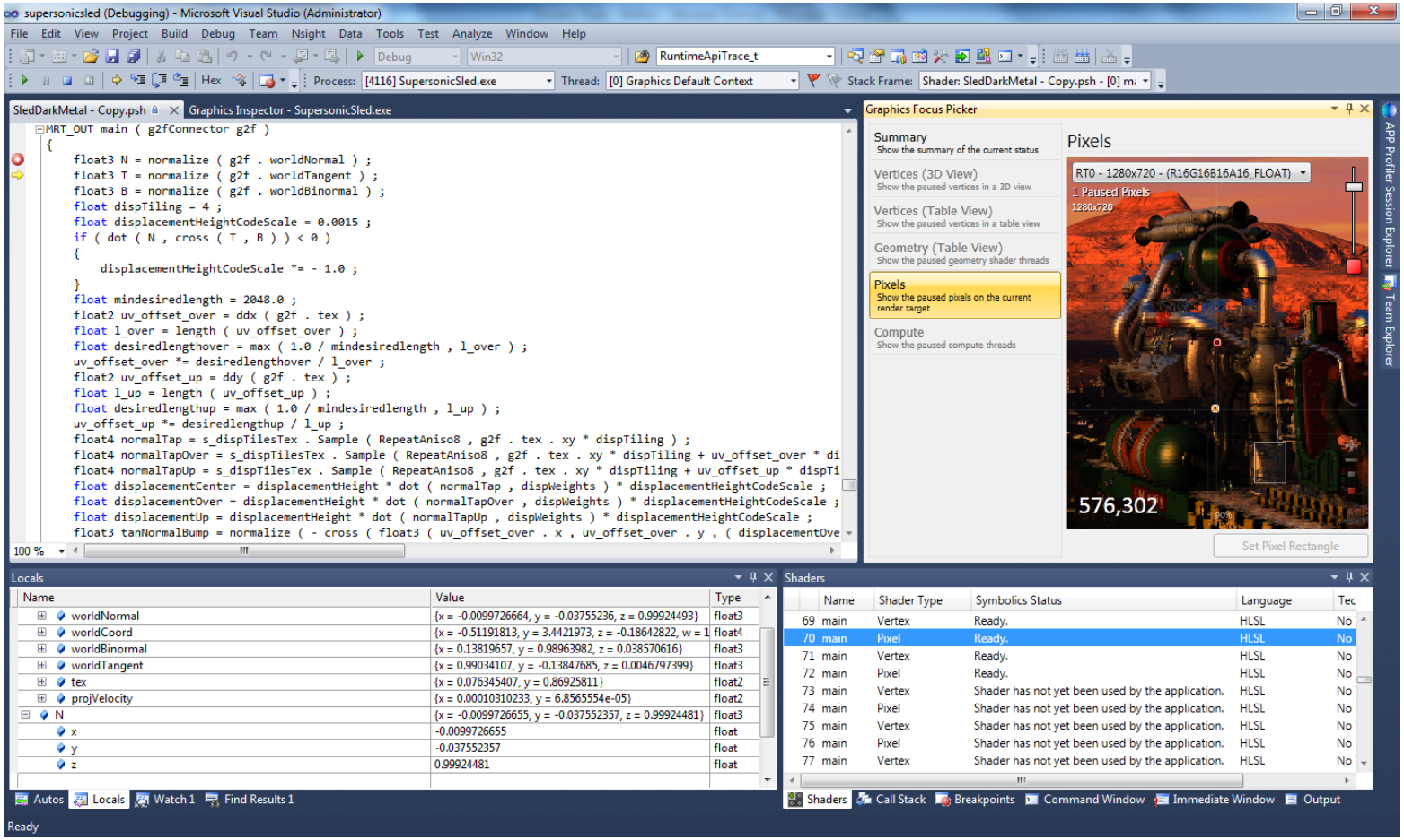
```
int attribList[] =
{
  WGL_CONTEXT_MAJOR_VERSION_ARB, 1,
  WGL_CONTEXT_MINOR_VERSION_ARB,  0,
  WGL_CONTEXT_FLAGS_ARB, WGL_CONTEXT_FORWARD_COMPATIBLE_BIT_ARB |
                         WGL_CONTEXT_DEBUG_BIT_ARB,

  0, 0, 0
};

attribList[1] = g_major;
attribList[3] = g_minor;

if (!(hRCTemp = wglCreateContextAttribsARB(g_hDC, 0, attribList)))
{
  destroyWindow();
  return false;
}
```

OpenGL Debug Output

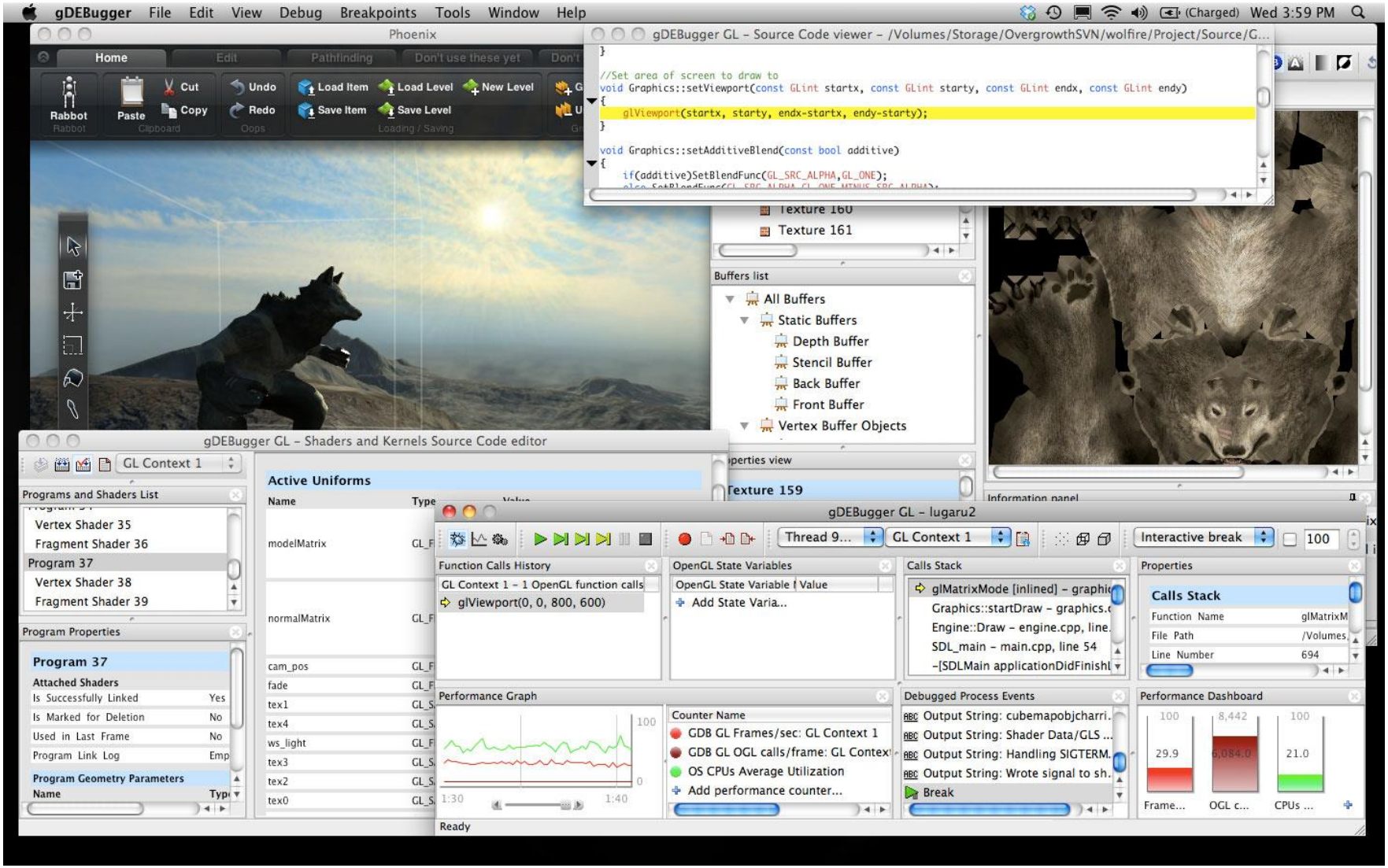
```
void CALLBACK DebugCallback(unsigned int source, unsigned int type,  unsigned int id, unsigned int severity,
                                        int length, const char* message, void* userParam)
{
  std::cerr << "GL_Error  : " << message << std::endl;
}

glEnable(GL_DEBUG_OUTPUT_SYNCHRONOUS_ARB);
glDebugMessageCallbackARB(&DebugCallback, NULL);

glDebugMessageControlARB(GL_DONT_CARE, GL_DONT_CARE, GL_DONT_CARE, 0, NULL, GL_FALSE);
glDebugMessageControlARB(GL_DONT_CARE, GL_DEBUG_TYPE_ERROR_ARB, GL_DONT_CARE, 0, NULL, GL_TRUE);
….
glDebugMessageControlARB(GL_DONT_CARE, GL_DEBUG_TYPE_PERFORMANCE_ARB, GL_DONT_CARE, 0, NULL, GL_TRUE);
glDebugMessageControlARB(GL_DONT_CARE, GL_DEBUG_TYPE_DEPRECATED_BEHAVIOR_ARB, GL_DONT_CARE,0,NULL, GL_TRUE);
glDebugMessageControlARB(GL_DONT_CARE, GL_DEBUG_TYPE_PORTABILITY_ARB, GL_DONT_CARE, 0, NULL, GL_TRUE);
glDebugMessageControlARB(GL_DONT_CARE, GL_DEBUG_TYPE_UNDEFINED_BEHAVIOR_ARB, GL_DONT_CARE, 0, NULL, GL_TRUE);
….

// ERROR. Debug output callback should be called at this point
//
glBindVertexArray(-1); CHECK_GL_ERRORS;
```

Отладка в OpenGL

- Nvidia Nsight

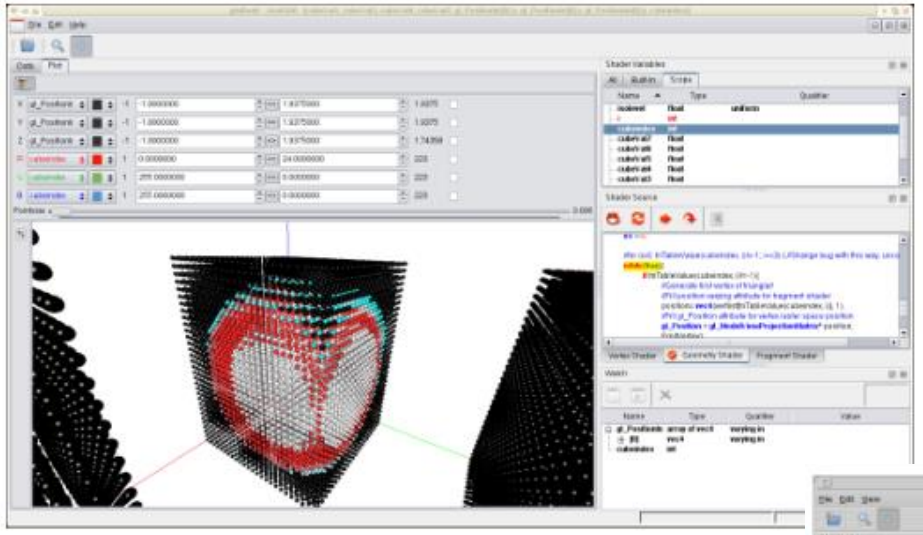


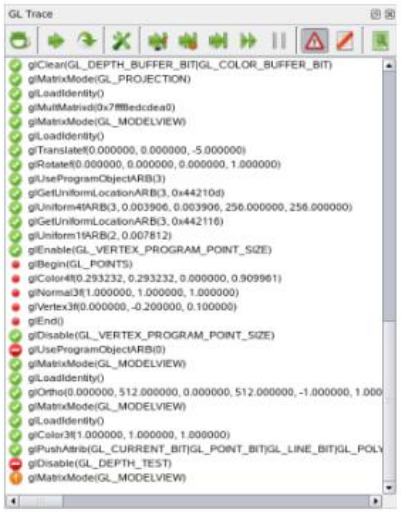
Отладка в OpenGL

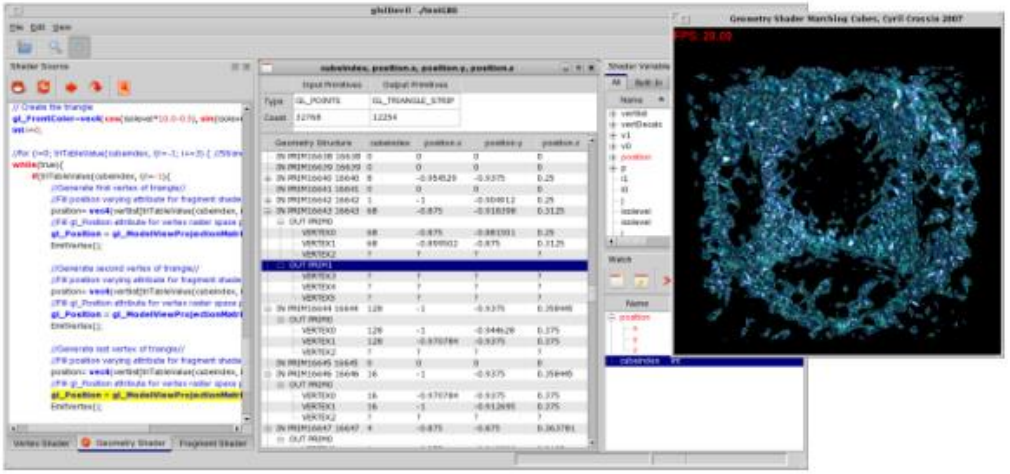


Отладка в OpenGL

- GLSL DevIL







Разработка сложных шейдеров

- DirectX11 “Dynamic Shader Linkage”
- OpenGL4 “Subroutine”
- Аналог виртуальных функций, но (!!!)
  - Это не виртуальные функции!

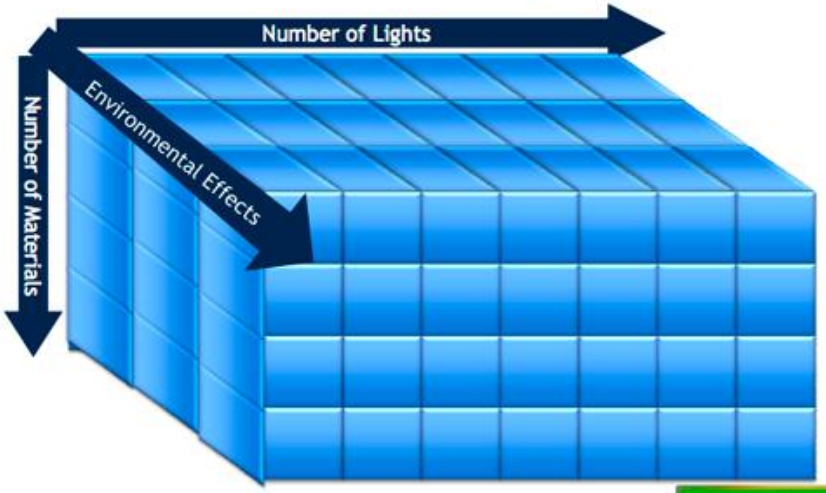
Über-shader

```
foo (...) {
    if (m == 1) {
        // do material 1
    } else if (m == 2) {
        // do material 2
    }
    if (l == 1) {
        // do light model 1
    } else if (l == 2) {
        // do light model 2
    }
}
```

Dynamic Subroutine

```
Material1(...) { ... }
Material2(...) { ... }
Light1(...) { ... }
Light2(...) { ... }

foo(...) {
    myMaterial.Evaluate(...);
    myLight.Evaluate(...);
}
```

Combinatorial Explosion



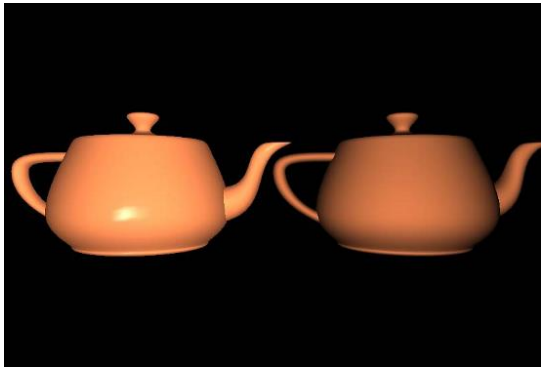
OpenGL subroutine

```
#version 400
subroutine vec3 shadeModelType( vec4 position, vec3 normal);
subroutine uniform shadeModelType shadeModel;

subroutine ( shadeModelType )
vec3 phongModel( vec4 position, vec3 norm )
{
  // The ADS shading calculations go here (see: "Using
  // functions in shaders," and "Implementing
  // per-vertex ambient, diffuse and specular (ADS) shading")
  …
}

subroutine ( shadeModelType )
vec3 diffuseOnly( vec4 position, vec3 norm )
{
  vec3 s = normalize( vec3(Light.Position - position) );
  return Light.Ld * Material.Kd * max( dot(s, norm), 0.0 );
}
```

OpenGL subroutine

```
subroutine vec3 shadeModelType( vec4 position, vec3 normal);
subroutine uniform shadeModelType shadeModel;

void main()
{
  vec3 eyeNorm;
  vec4 eyePosition;

  // Get the position and normal in eye space
  getEyeSpace(eyeNorm, eyePosition);

  // Evaluate the shading equation. This will call one of
  // the functions: diffuseOnly or phongModel
  LightIntensity = shadeModel( eyePosition, eyeNorm );

  gl_Position = MVP * vec4(VertexPosition,1.0);
}
```

OpenGL subroutine



```
glUseProgram (programHandle);

GLuint adsIndex =
glGetSubroutineIndex(programHandle, GL_VERTEX_SHADER,"phongModel");

GLuint diffuseIndex =
glGetSubroutineIndex(programHandle, GL_VERTEX_SHADER, "diffuseOnly");



glUniformSubroutinesuiv( GL_VERTEX_SHADER, 1, &adsIndex);
... // Render the left teapot

glUniformSubroutinesuiv( GL_VERTEX_SHADER, 1, &diffuseIndex);
... // Render the right teapot
```

Ссылки

- OpenGL 3.0+/4.0+
  - http://steps3d.narod.ru
  - http://www.gamedev.ru/community/ogl/
  - http://nopper.tv/opengl.html
  - http://www.arcsynthesis.org/gltut/
  - http://www.opengl-tutorial.org/
  - http://www.opengl.org/wiki
  - http://www.opengl.org/sdk/docs/man3/
  - http://www.opengl.org/wiki/Common_Mistakes
- OpenGL 4.0+
  - http://www.opengl.org/sdk/docs/man4/
  - http://www.nvidia.com/content/devzone/opengl-driver-4.3.html

Вопросы на внимательность

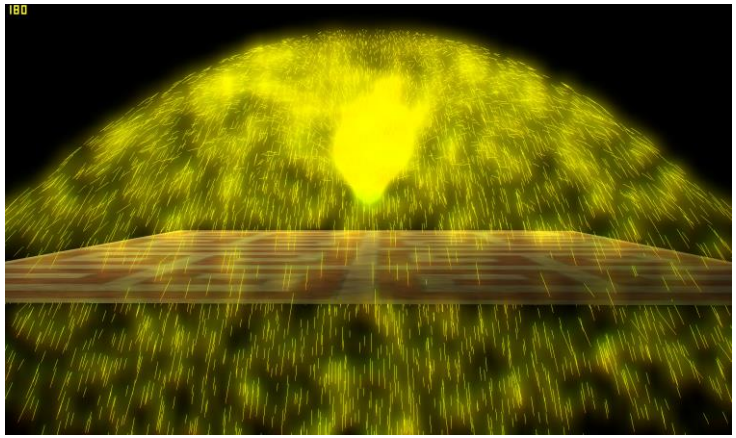
- В чем особенность работы с объектами в OpenGL ?

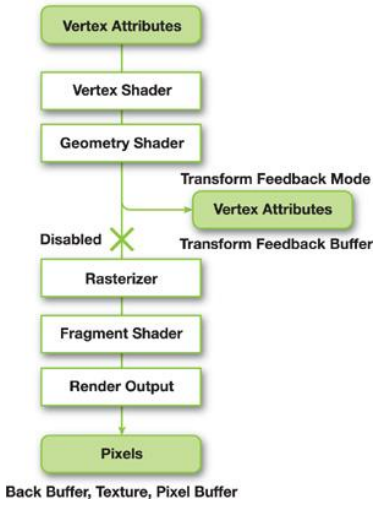
```
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, N*sizeof(GLfloat), (GLfloat*)cpuData, GL_STATIC_DRAW);
```

- Что добавляет расшинение EXT_direct_state_access?

```
glNamedBufferDataEXT (vbo, N*sizeof(GLfloat), (GLfloat*)cpuData, GL_STATIC_DRAW);
```

- Для чего нужен Transform Feedback?
  - В чем особенность?





Вопросы на внимательность

- В чем отличие gl4 subroutine от виртуальных функций?

  Статическая диспетчеризация, управляемая со стороны CPU

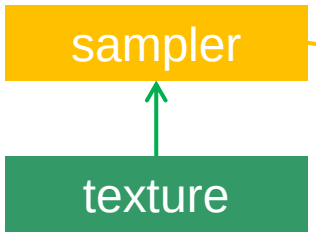
- Как ловить ошибки OpenGL ?

  Макрос CHECK_GL_ERROR после каждого вызова +
  Debug Call Back для получения детальной информации об ошибке

- В чем отличие привязки текстур в GL 3 и 4 ?

  В OpenGL4 введен объект Sampler

```
glSamplerParameteri (s_id, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glSamplerParameteri (s_id, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
```




```
void main(void)
{
  vec4 color2 = texture(someSampler, textureCoords);
  …
}
```

Вопросы на внимательность

- Что такое трилинейная фильтрация?

  Билинейная фильтрация на 2 ближайших мип уровнях +
  линейная фильтрация между ними

- В чем отличие в поведении функций

  glBindBuffer(…); Буфер может быть привязан к различным слотам “target”
  glBindTexture(…); Первый вызов этой ф-ии раз и навсегда определяет тип текстуры

- Сколько треугольников будет нарисовано ?

  glDrawArrays(GL_TRIANGLE_STRIP 0, 6);