Современный графический конвейер и программный интерфейс OpenGL 4

Владимир Фролов (vfrolov@graphics.cs.msu.ru)
К.ф.-м.н., ВМиК МГУ

Компьютерная графика в играх

Компьютерная графика в играх

Компьютерная графика в играх

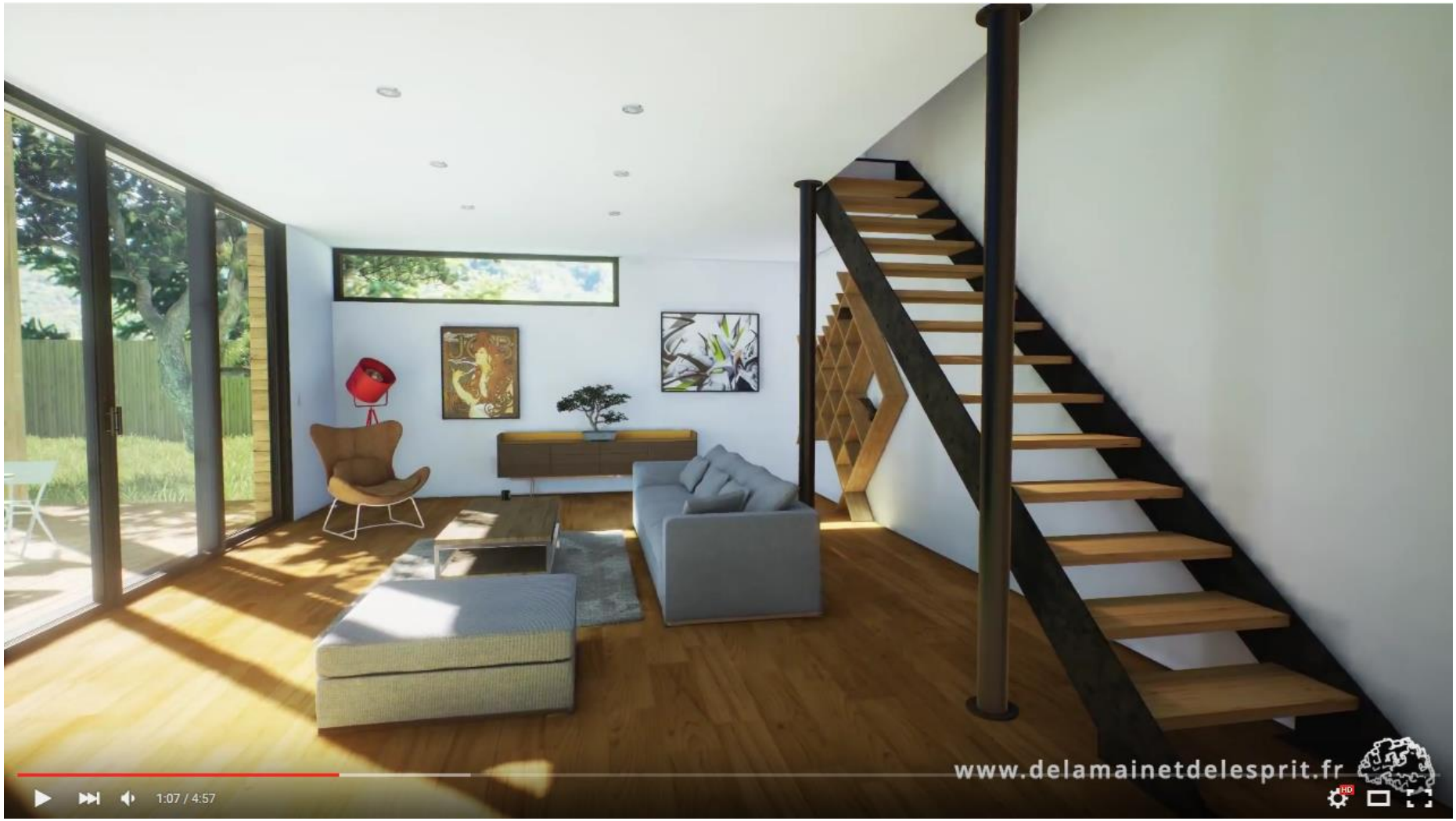
Компьютерная графика в играх

Компьютерная графика в играх

Компьютерная графика в играх

Компьютерная графика в презентациях

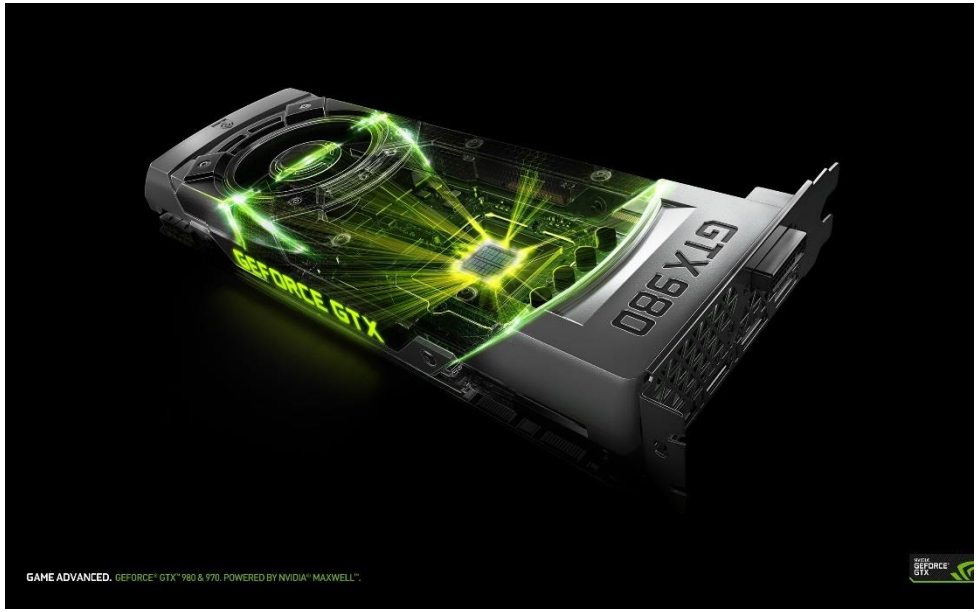
Компьютерная графика в презентациях

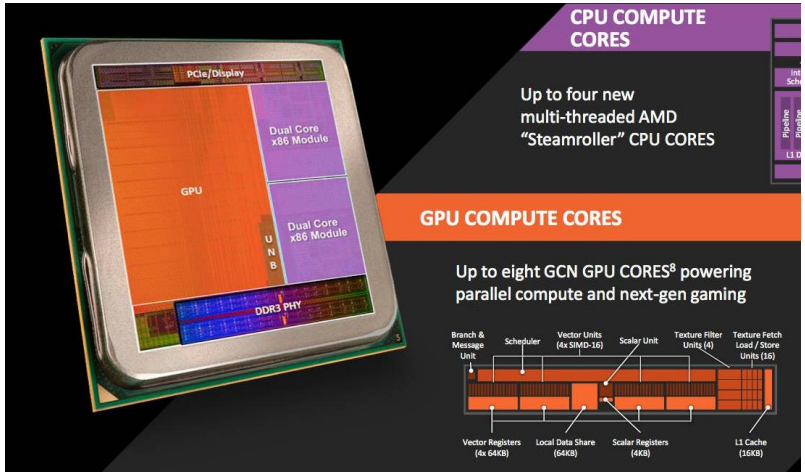


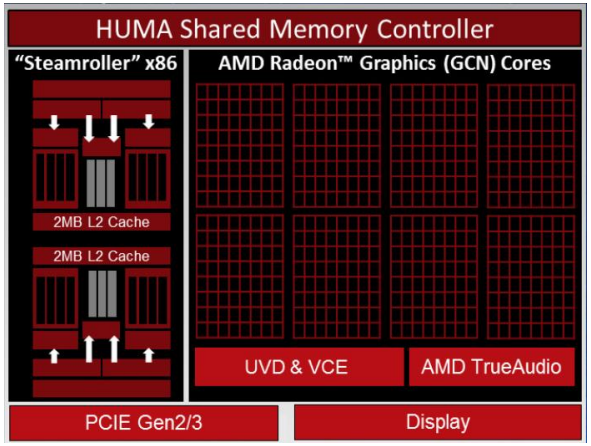
Компьютерная графика в презентациях

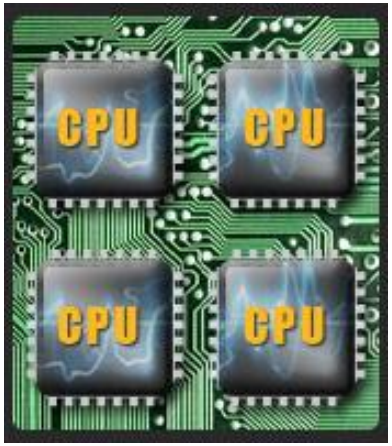
Что для этого нужно?

Что для этого нужно?





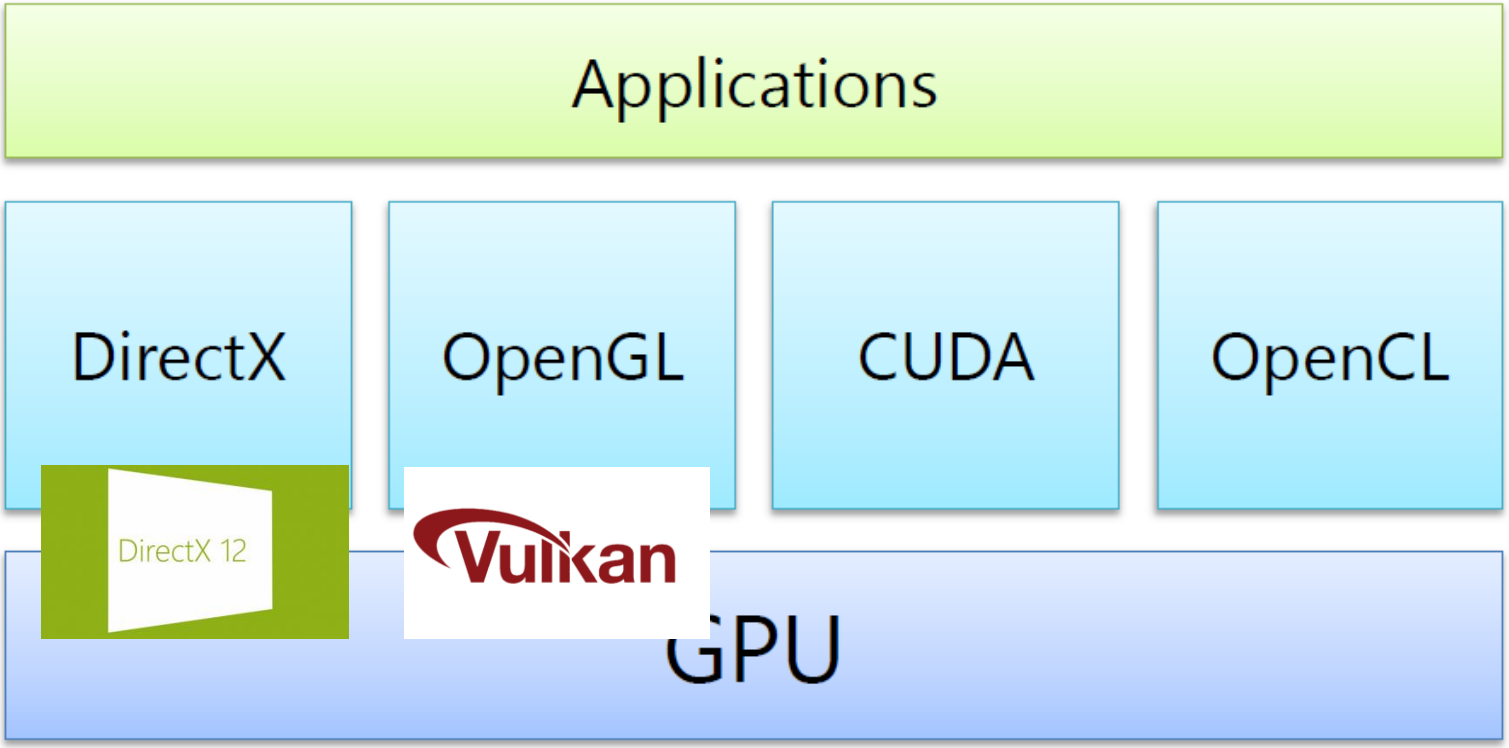




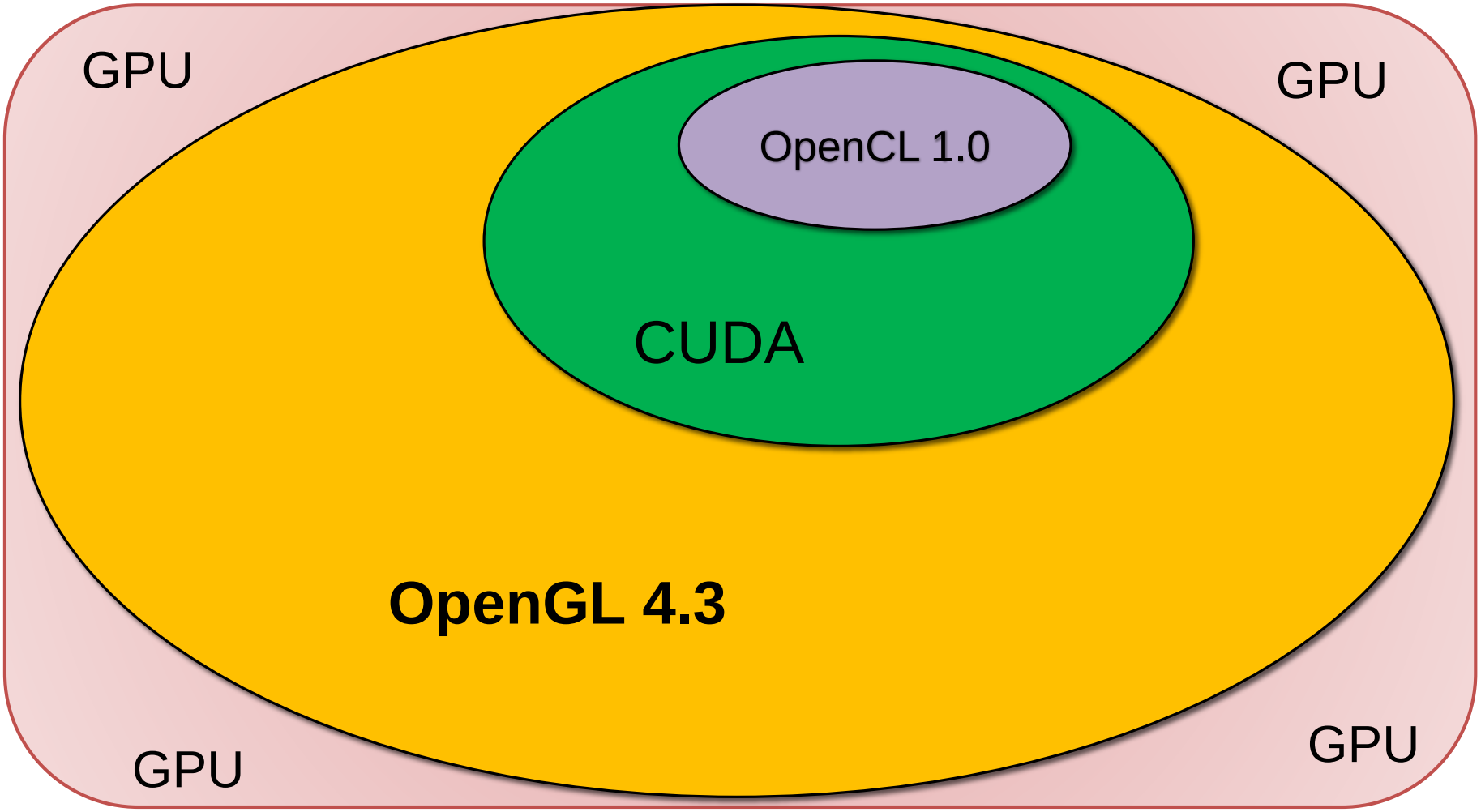
Что для этого нужно?

API программирования GPU

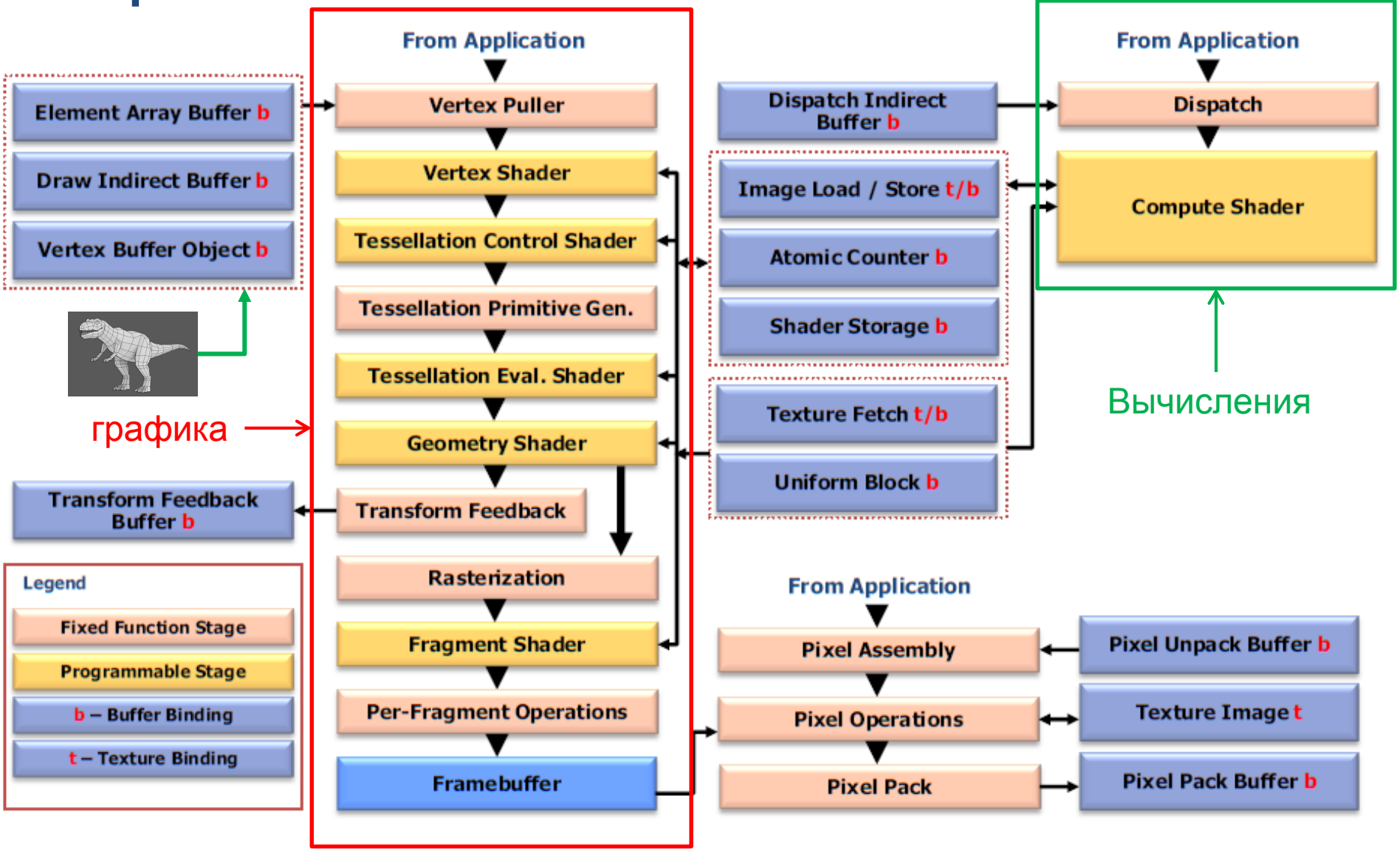
- Графика
- Вычисления общего назначения



Доступ к функциональности GPU



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
графика
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
Legend
Fixed Function Stage
Programmable Stage
b – Buffer Binding
t – Texture Binding

Модели

Рис. 7.3 Идеальная модель

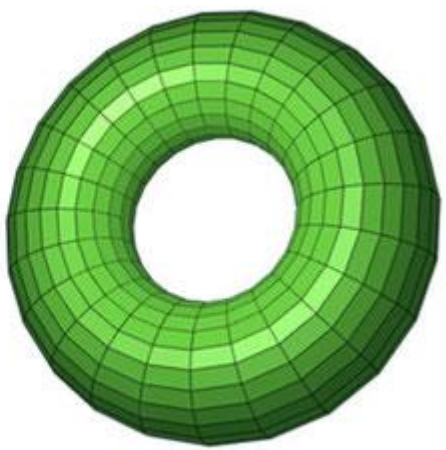
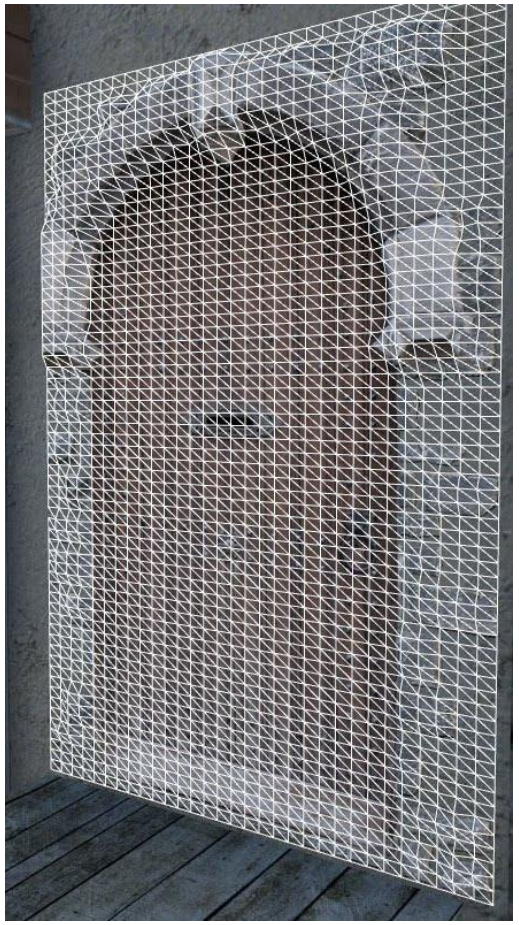
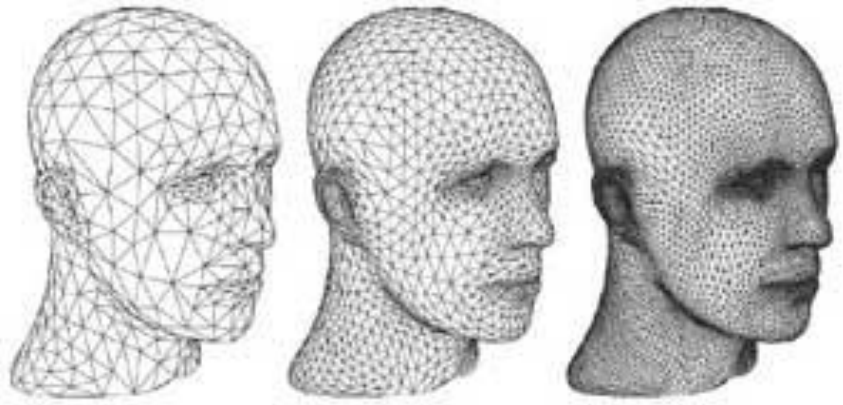
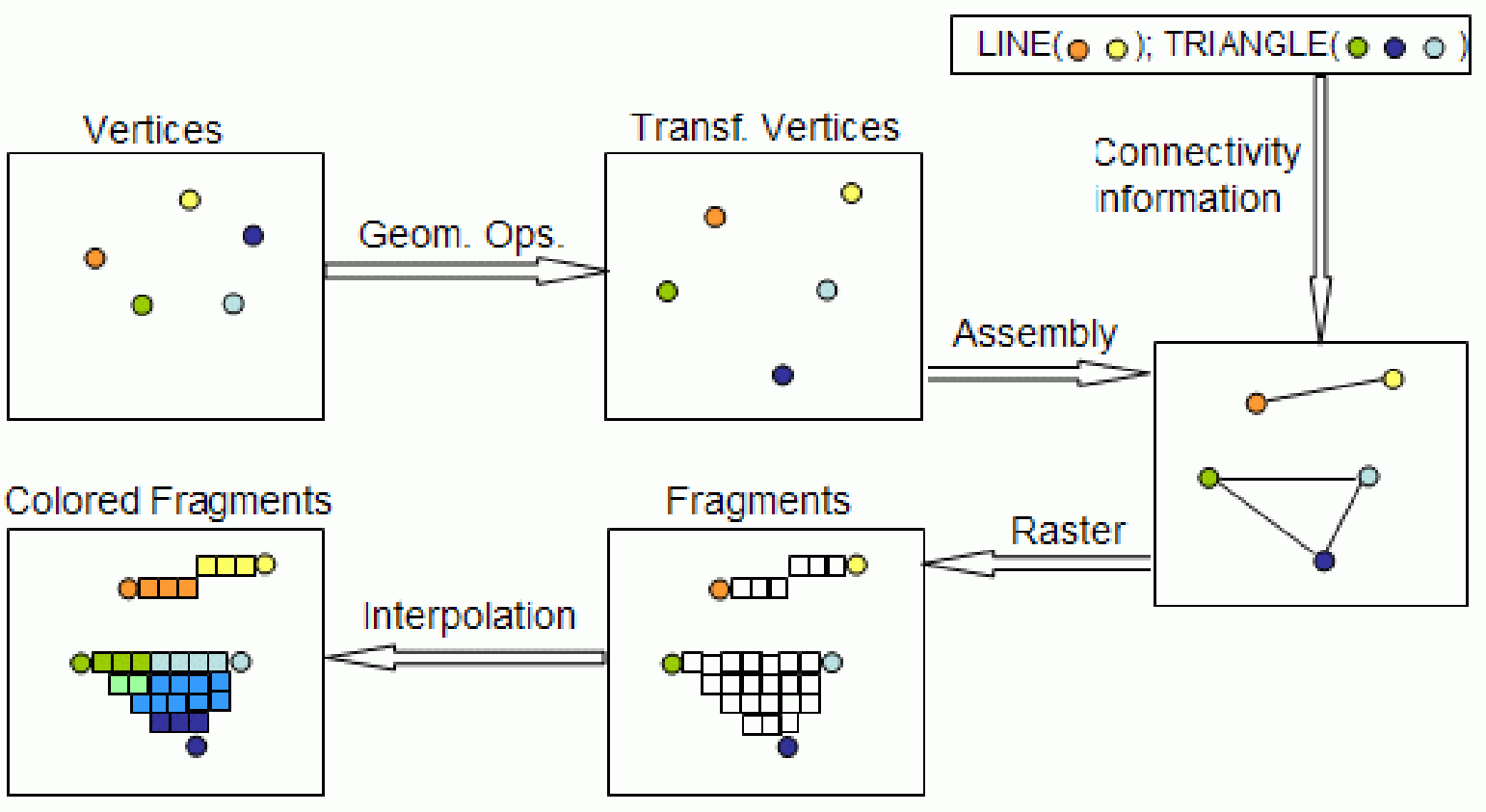


Рис. 7.3 Полигональное приближение

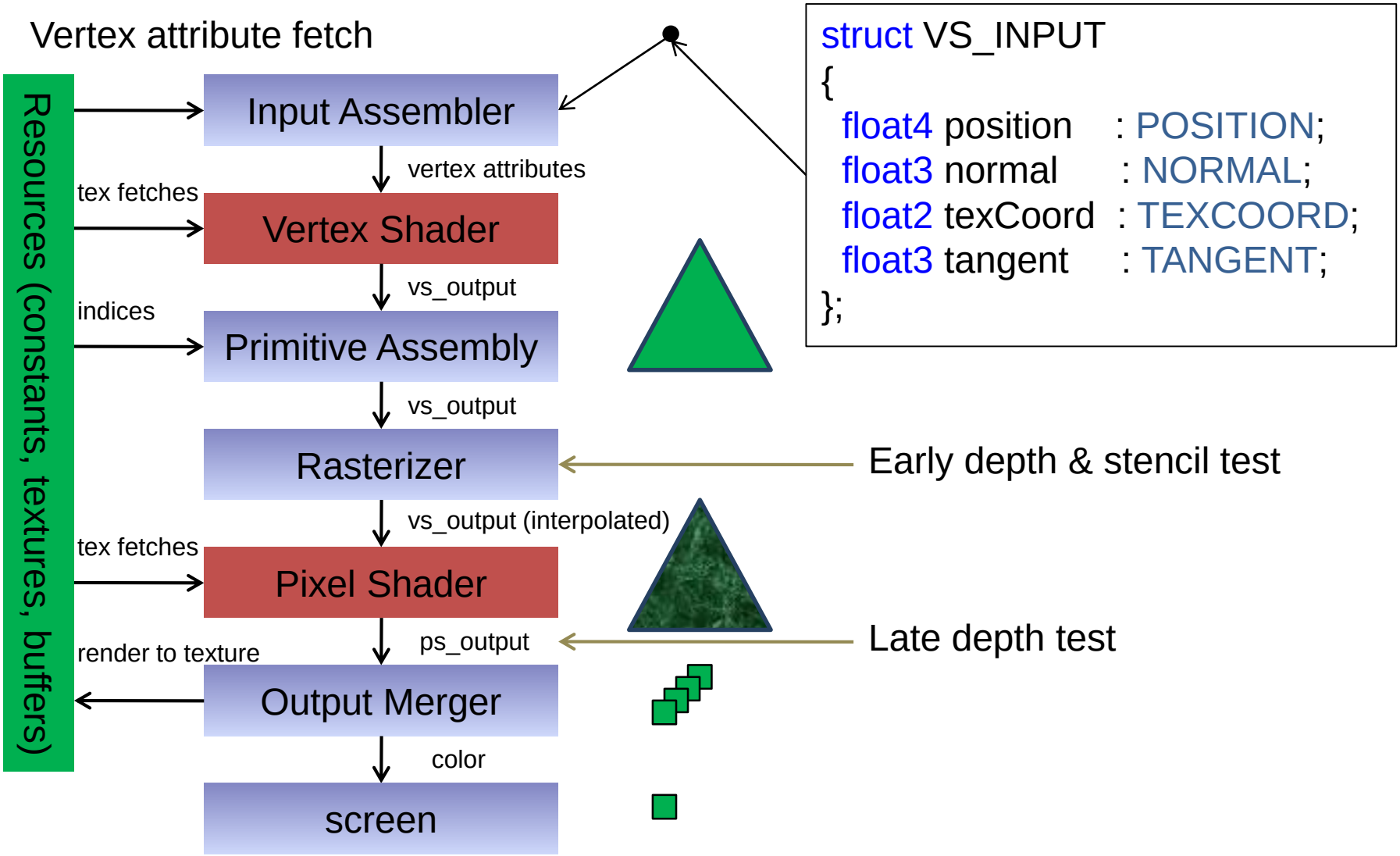




Графический конвейер (упрощённо)

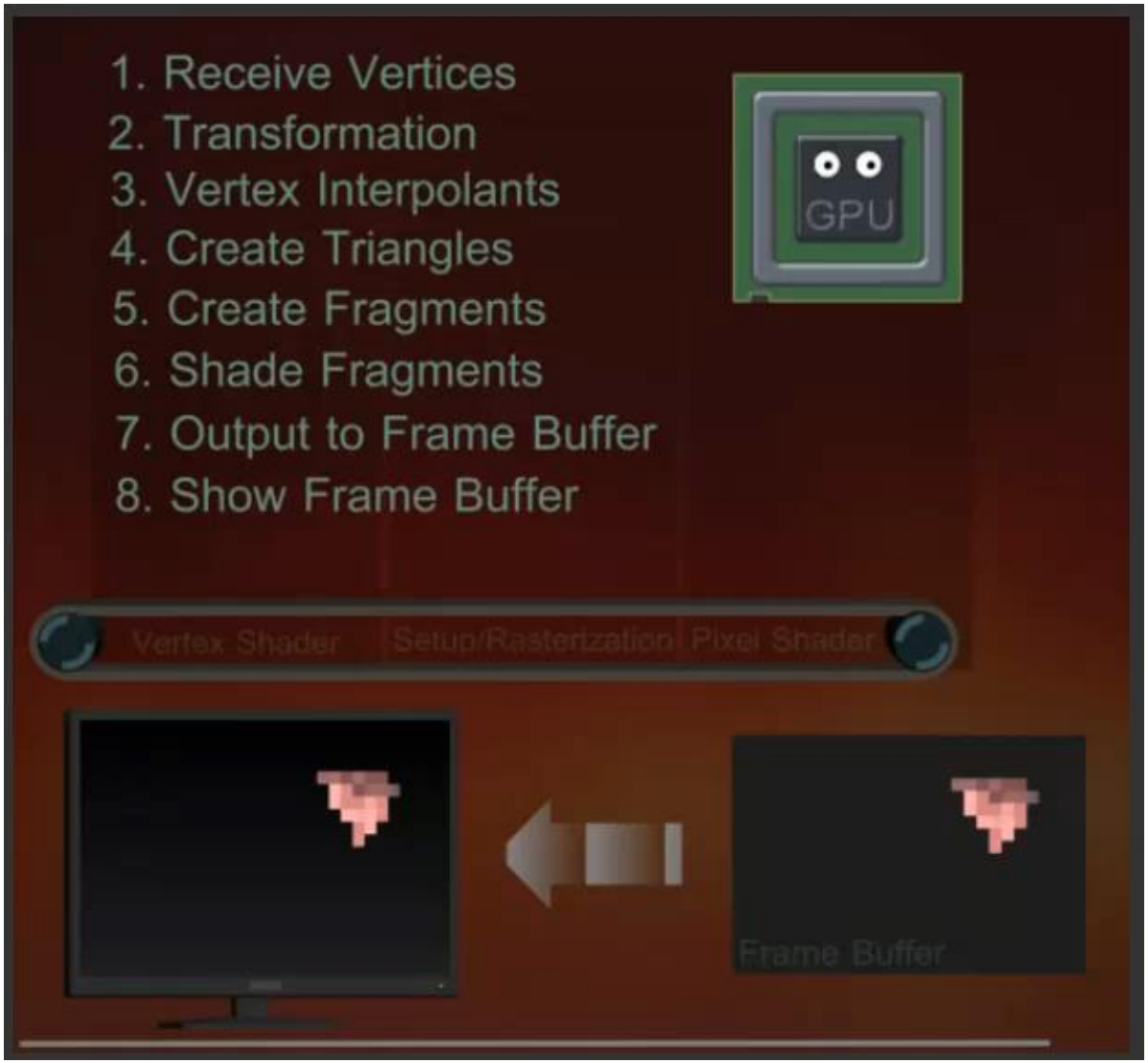


Графический конвейер и шейдеры

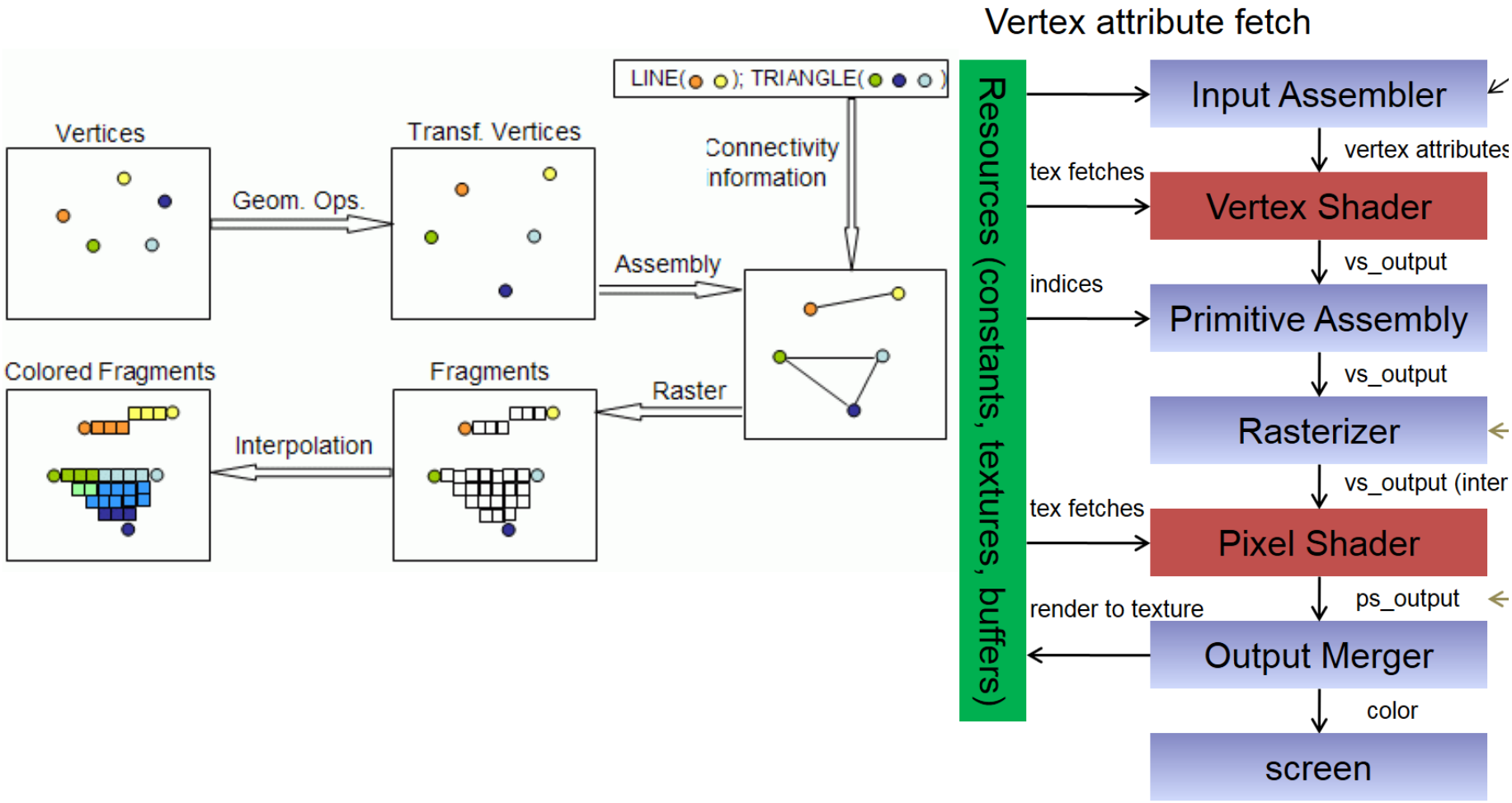


Графический конвейер

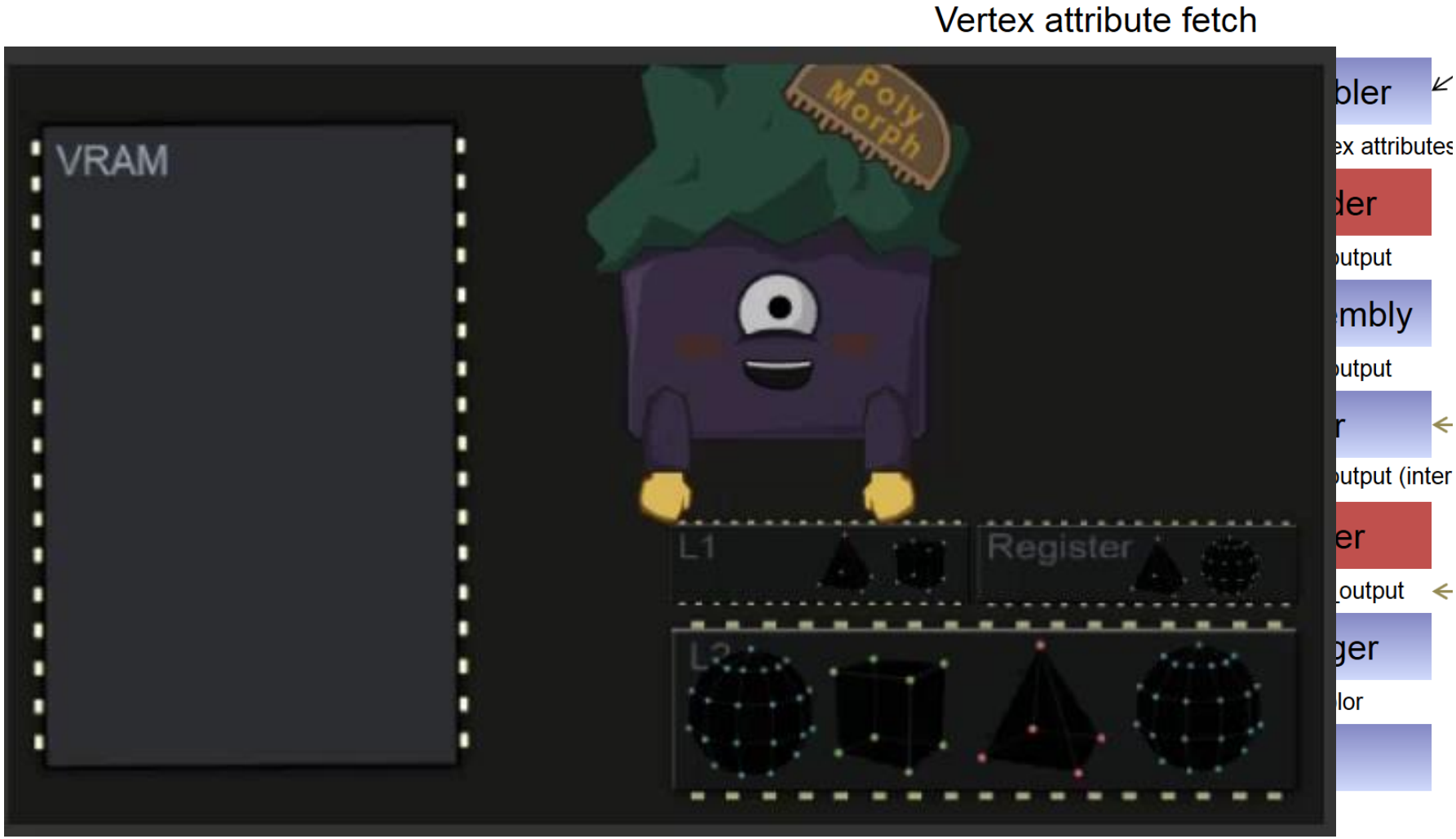
Графический конвейер



Графический конвейер



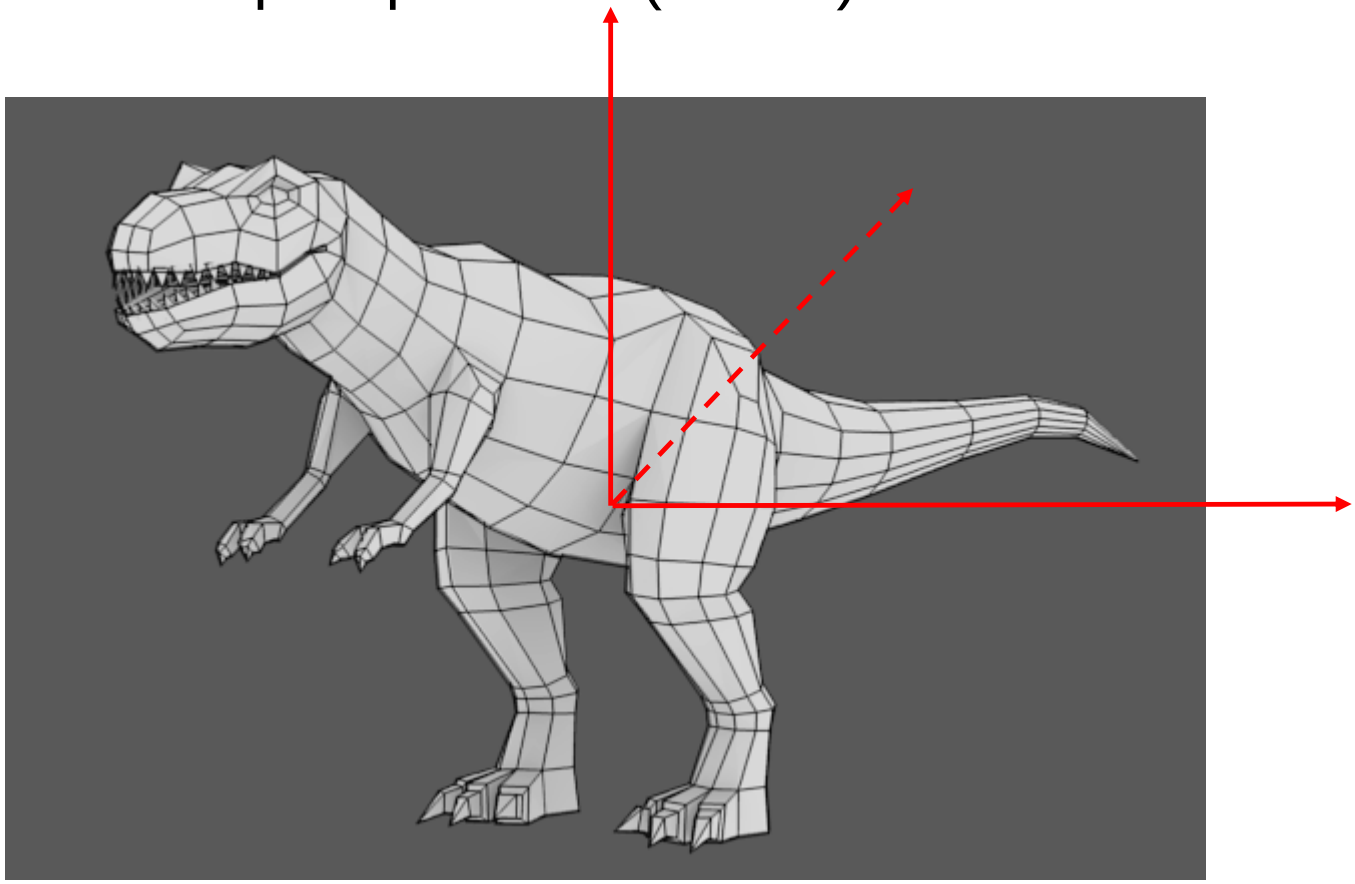
Графический конвейер



https://simonschreibt.de/gat/renderhell-book1/

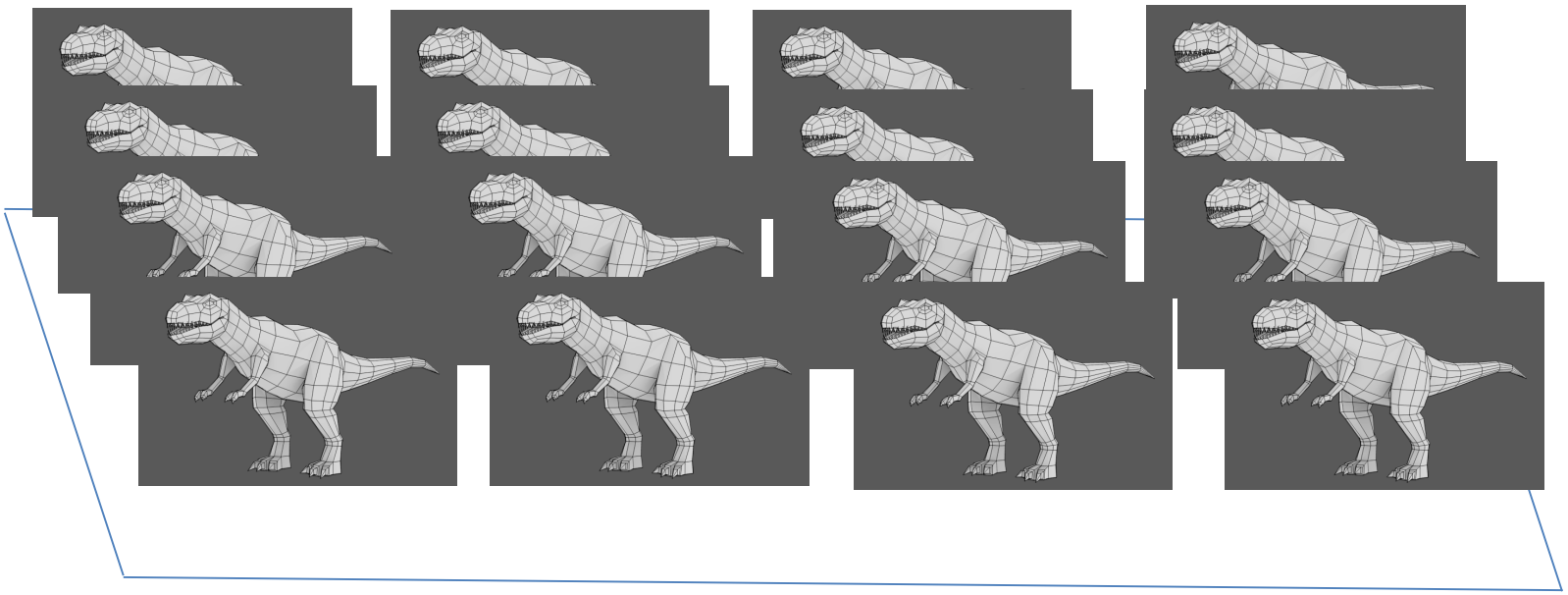
Model => World =>View => Clip

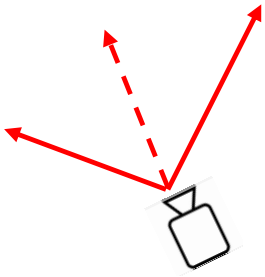
- Модельное пространство (Model)



Model => **World** =>View => Clip

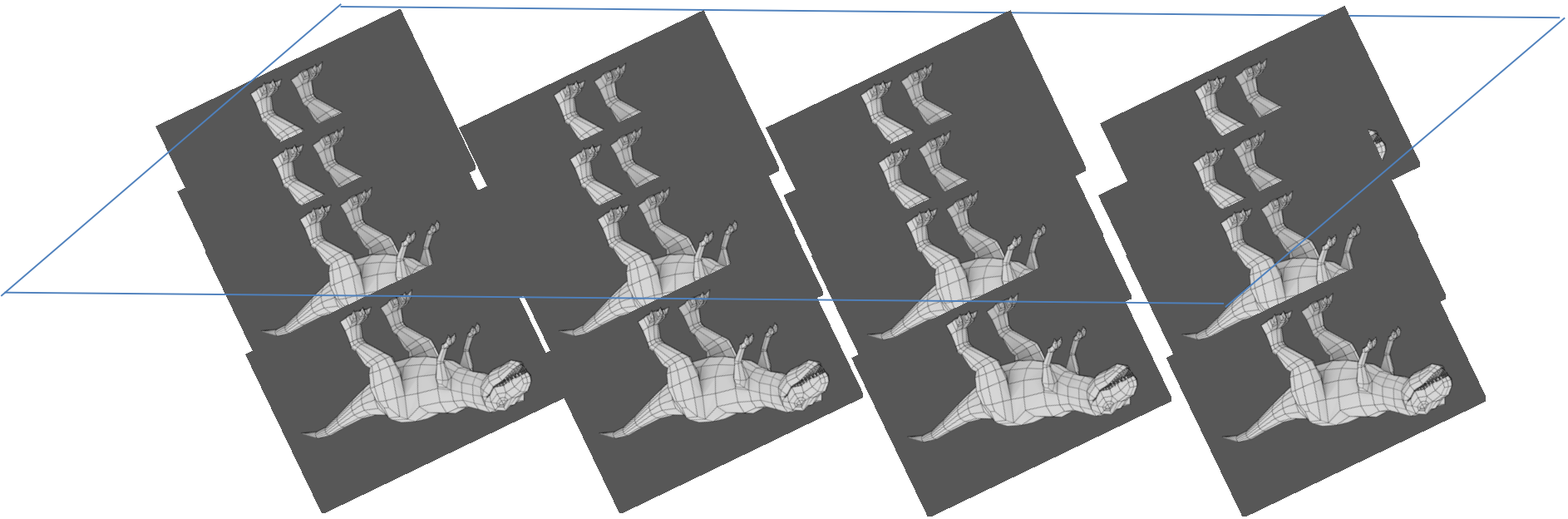
- Мировое пространство (World)

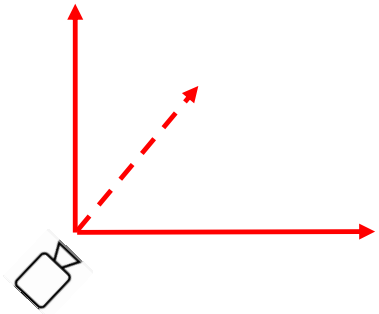




Model => World =>**View** => Clip

- Пространство камеры (World)

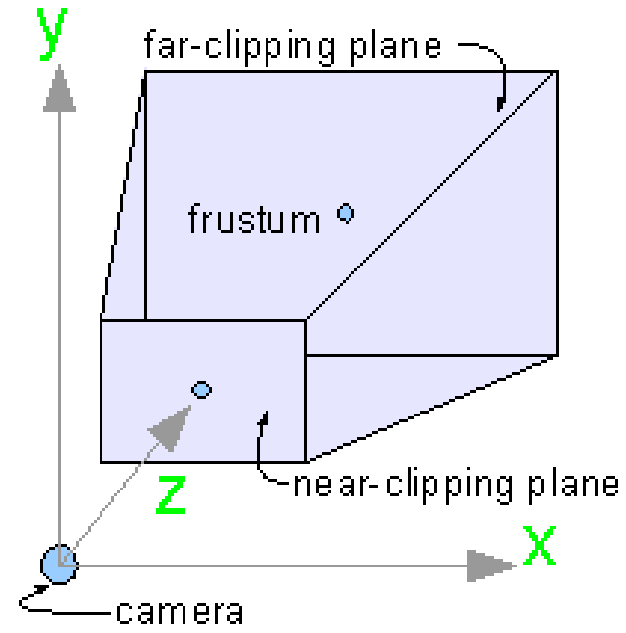




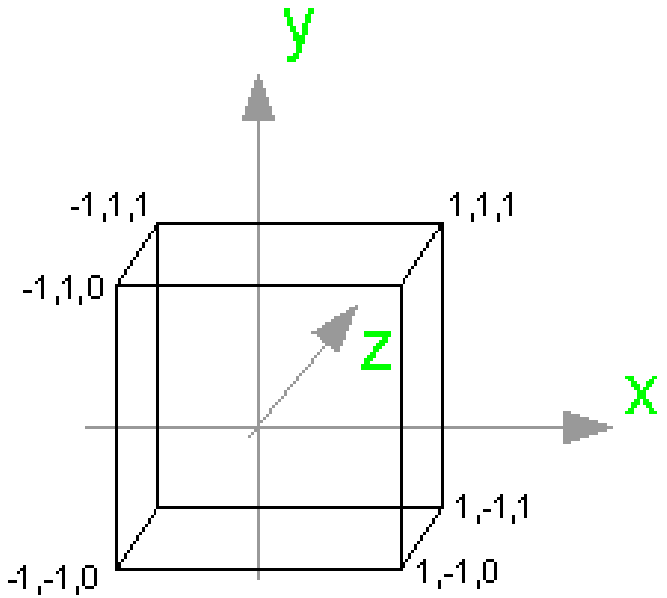
Камера находится в (0,0,0) и смотрит по оси Z (0,0,1)

Model => World =>View => **Clip**

- На выходе из вершинного шейдера коорд. в clip space



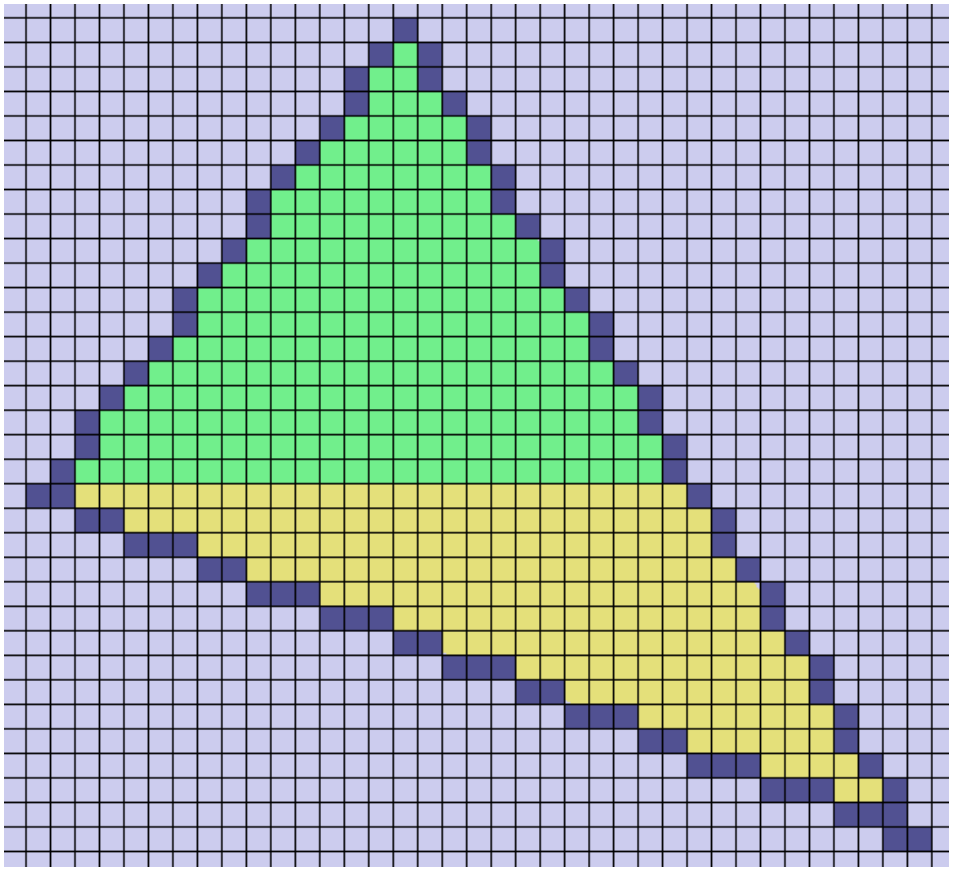
Camera Space

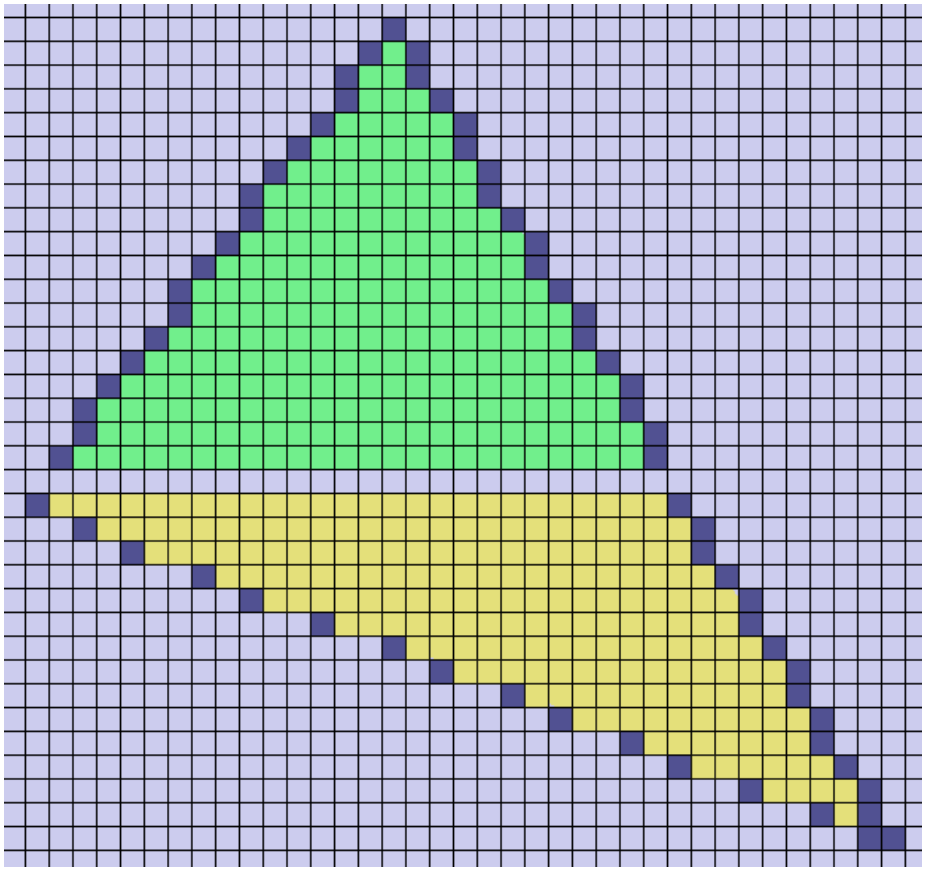


Clipping Space

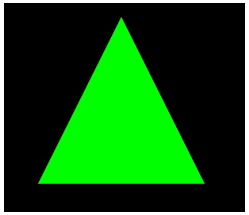
Растеризация

- Сама растеризация не сложнее закраски прямоугольника!

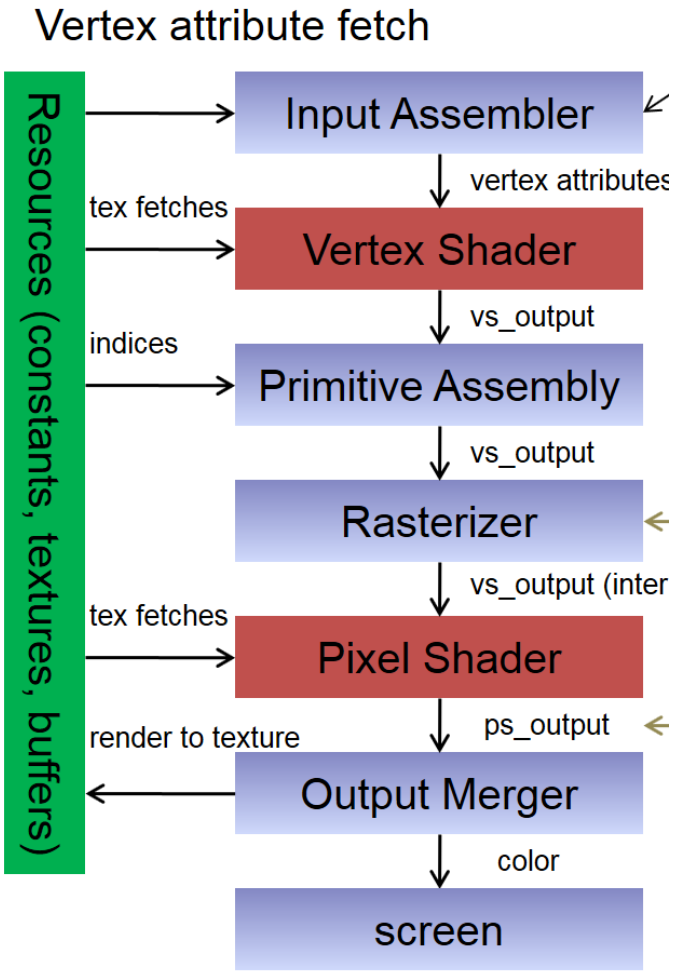




OpenGL3

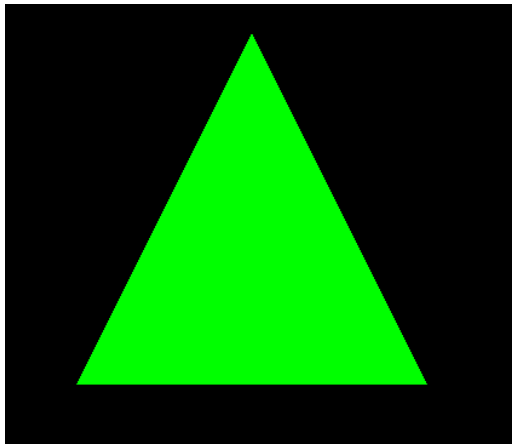


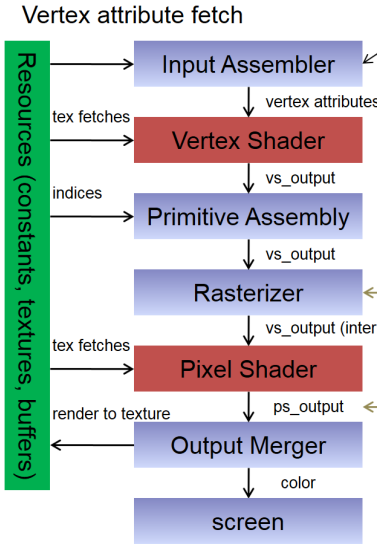
- Рисуем треугольник
  - Загрузка текста шейдеров из файла
  - Компиляция (vs_text, ps_text) -> (vs, fs)
  - Линковка (vs, fs) => program
    - g_prog = ShaderProgram(…);
  - Создание и загрузка данных
    - GLuint g_vbo1 = … // загружаем позиции вершин
    - GLuint g_vbo2 = … // на GPU
    - (g_vbo1, 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



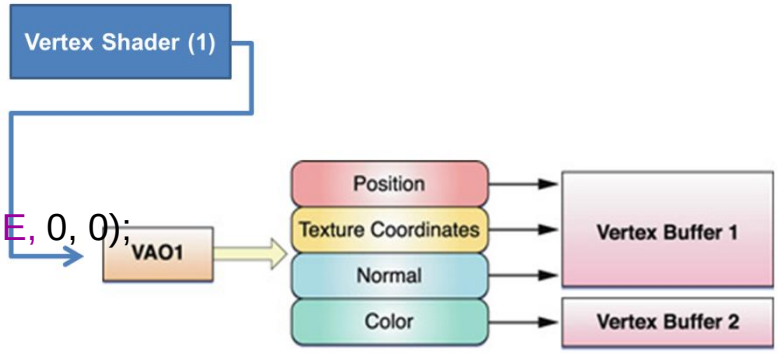
OpenGL3 (1 - VBO, VAO)

```
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;
```

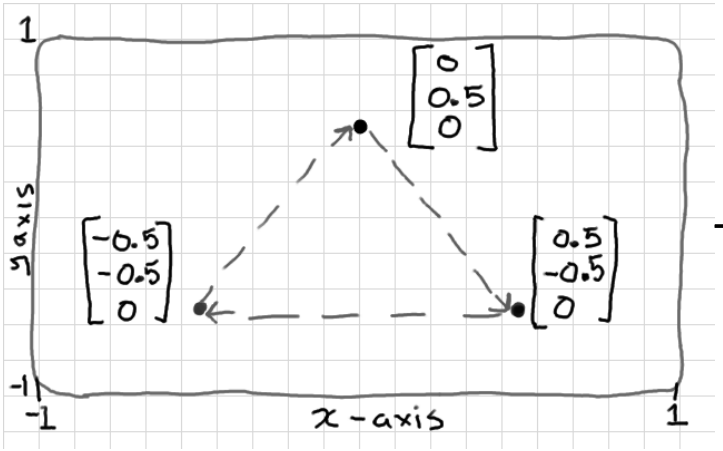





```
glGenBuffers(1, &g_vertexBufferObject);
glBindBuffer(GL_ARRAY_BUFFER, g_vertexBufferObject);
glBufferData(GL_ARRAY_BUFFER, 3*2*sizeof(GLfloat), (GLfloat*)trianglePos, GL_STATIC_DRAW);

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

glBindBuffer(GL_ARRAY_BUFFER, g_vertexBufferObject);
glEnableVertexAttribArray(vertexLocation);
glVertexAttribPointer(vertexLocation, 2, GL_FLOAT, GL_FALSE, 0, 0);
```



VBO и VAO



glBindBuffer(…);
glBufferData(…);

glBindBuffer(…);
glBufferData(…);

VBO (pos)

VBO (norm)

VAO

VS

Vertex attribute fetch

Resources (constants, textures, buffers)

Input Assembler

vertex attributes

tex fetches

Vertex Shader

vs_output

indices

Primitive Assembly

vs_output

Rasterizer

vs_output (inter

tex fetches

Pixel Shader

ps_output

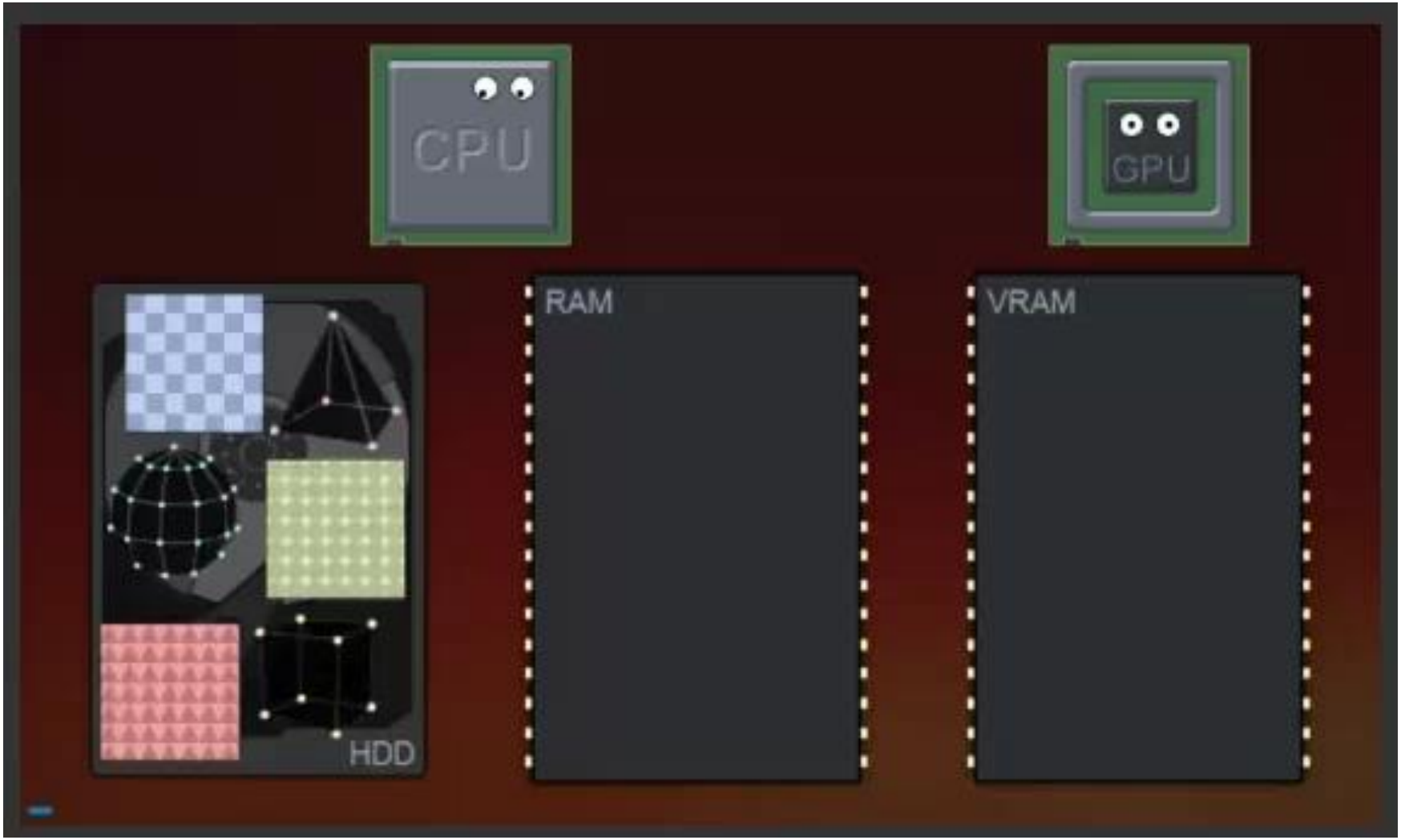
render to texture

Output Merger

color

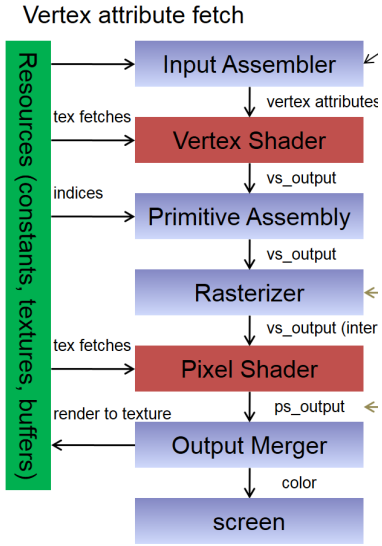
screen

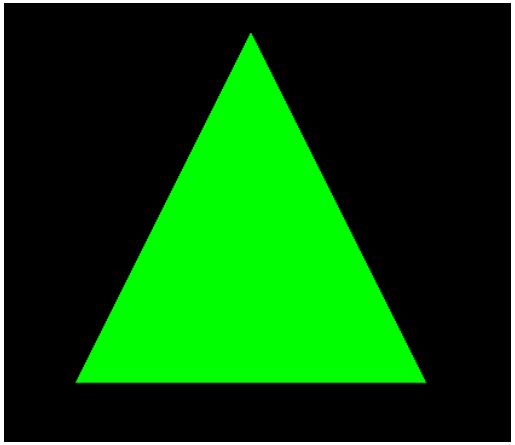
VBO



OpenGL3 (2 – Shader setup and Draw)

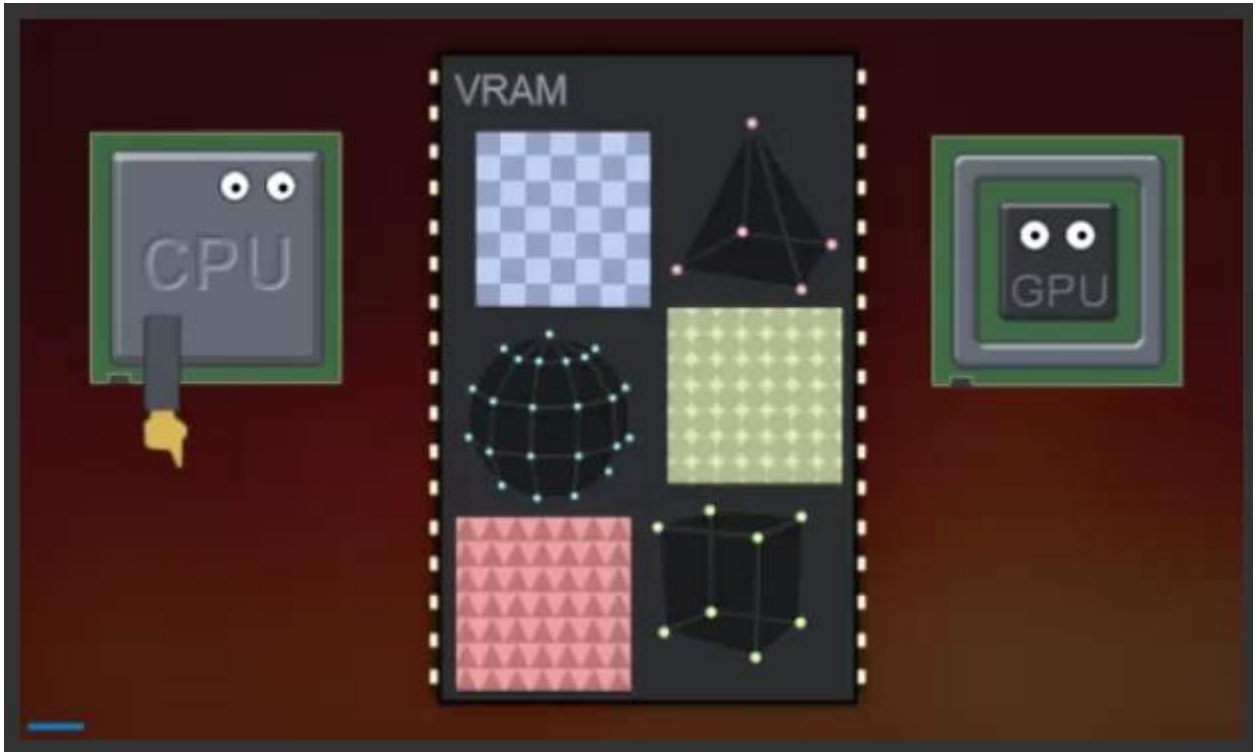
```
float model[16];
float modelView[16];

 // … расчёт матрицы  modelView


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

// … то же самое для матрицы проекции

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

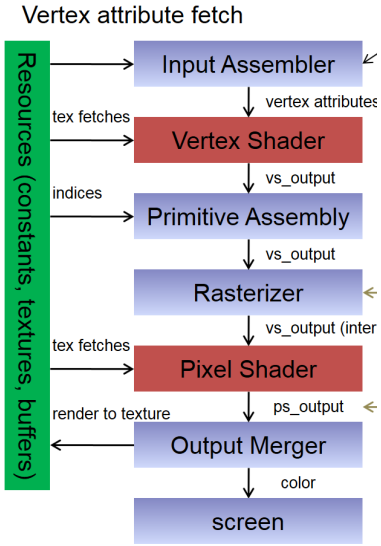




OpenGL3 (2 – Shader setup and Draw)

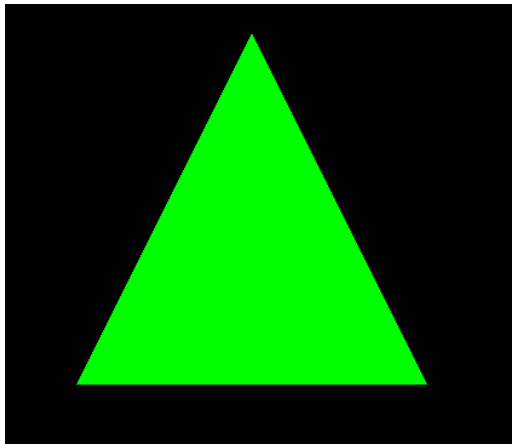
```
glBindVertexArray(g_vertexArrayObject);
glDrawArrays(GL_TRIANGLES, 0, 3);
```

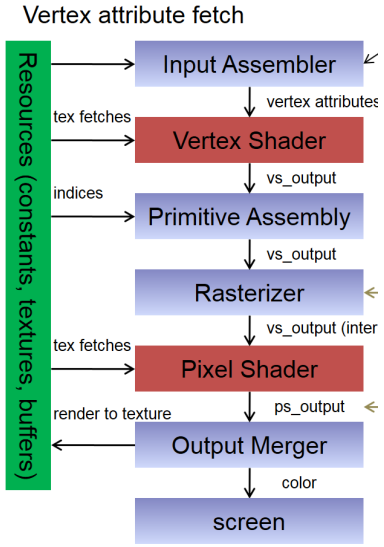


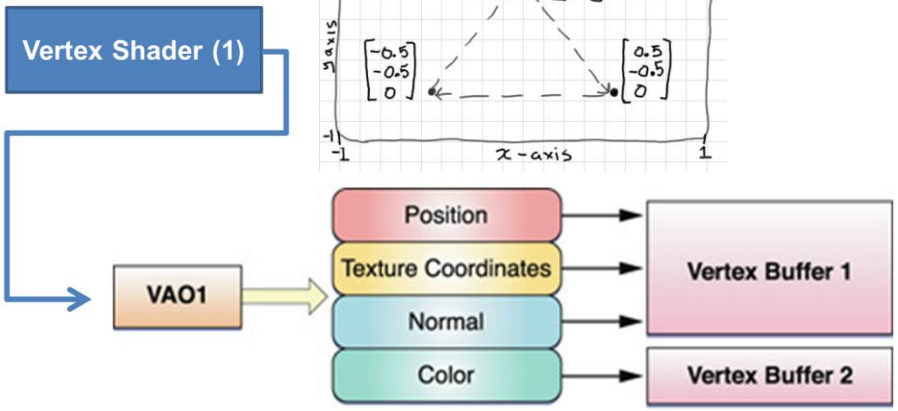


OpenGL3 (3 –Shaders)

```
// 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);
}
```

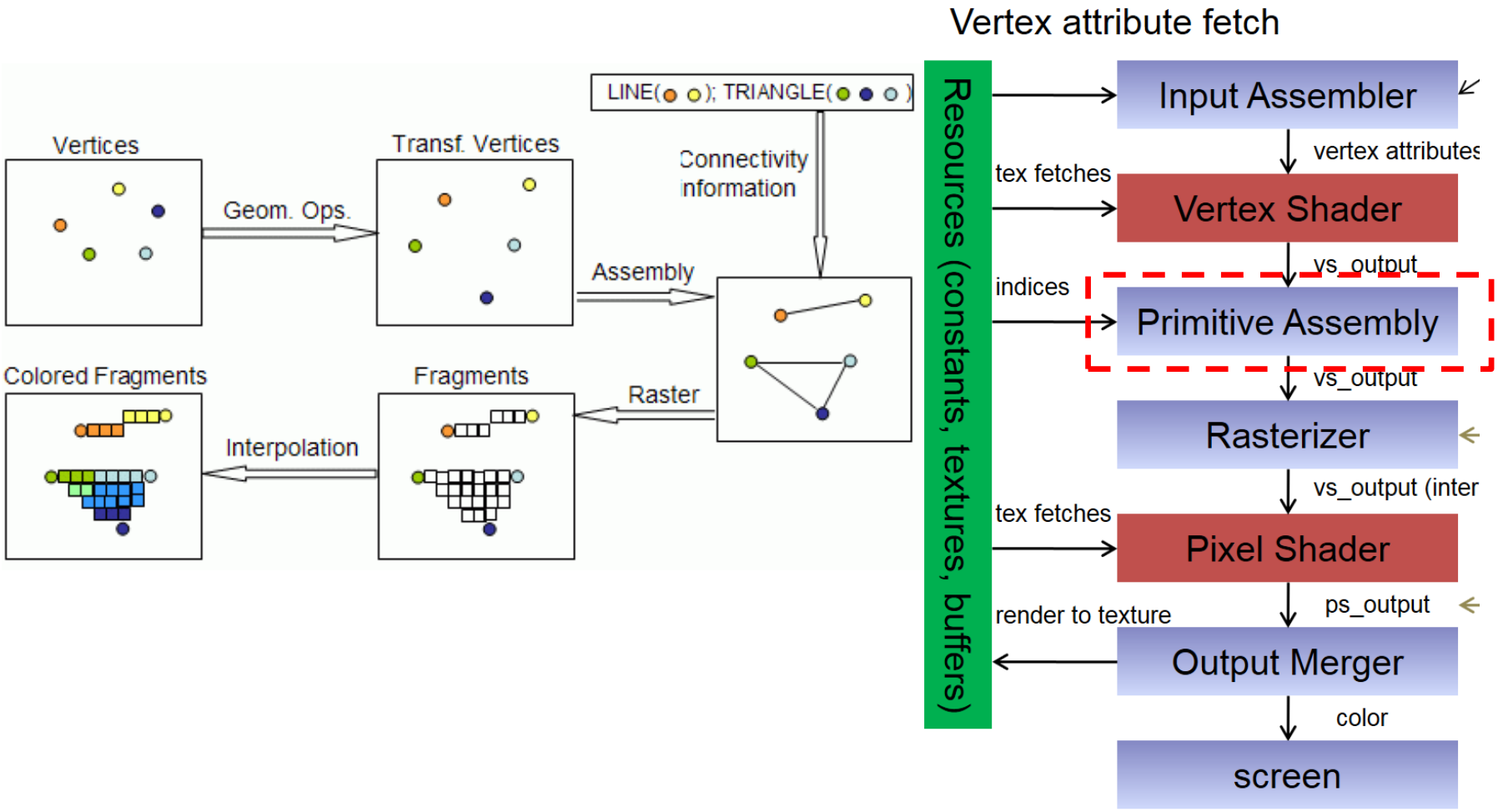






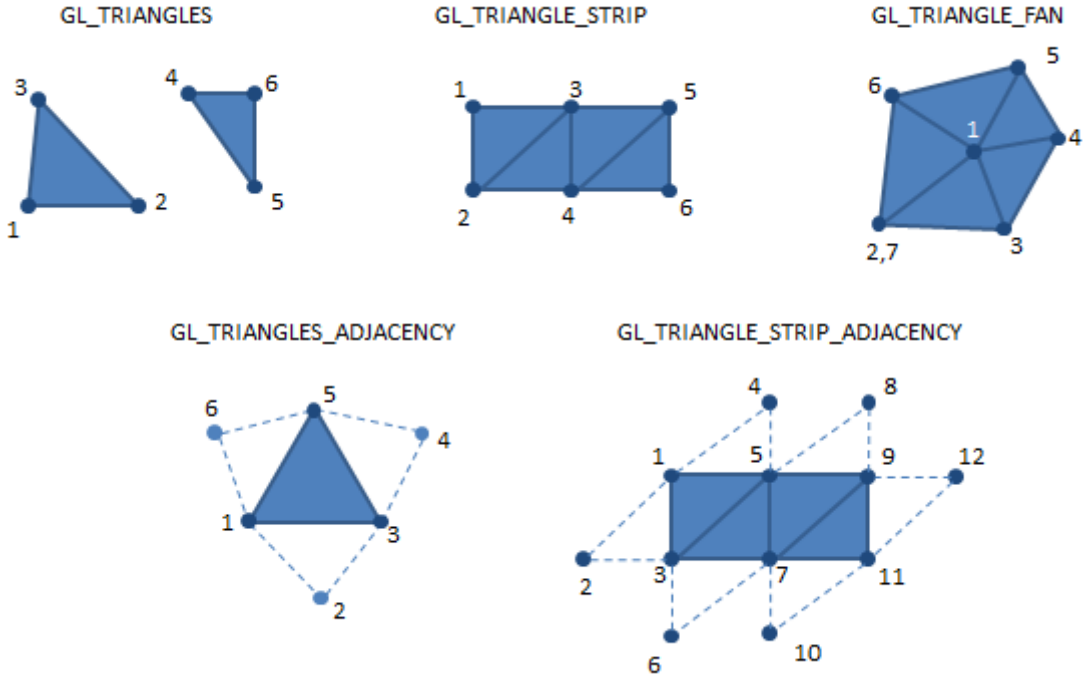
Сборка примитивов

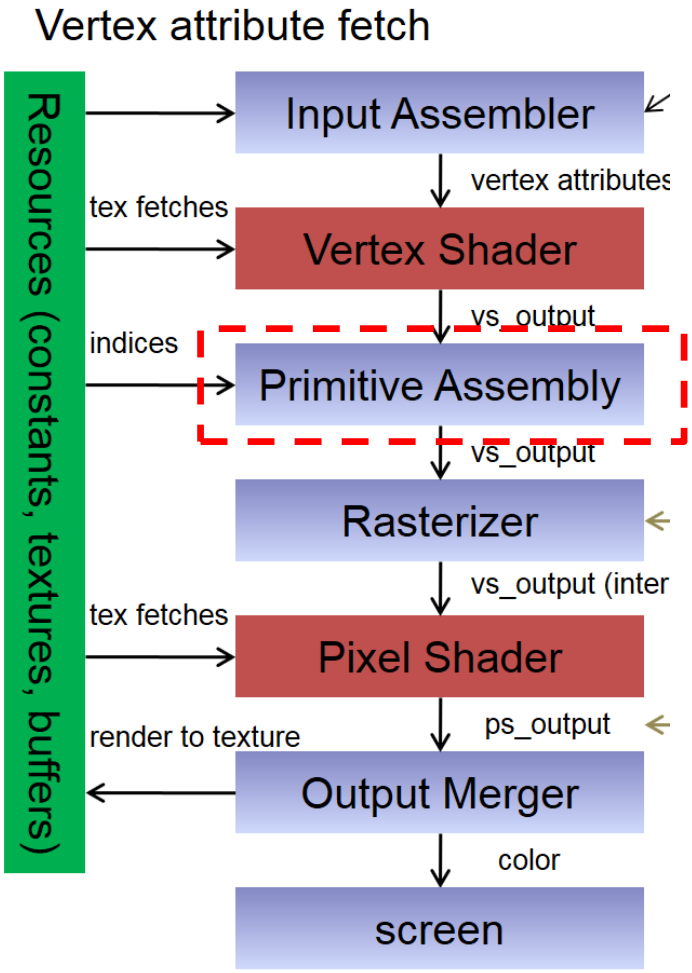
- В аппаратуре всё делается через треугольники



Сборка примитивов

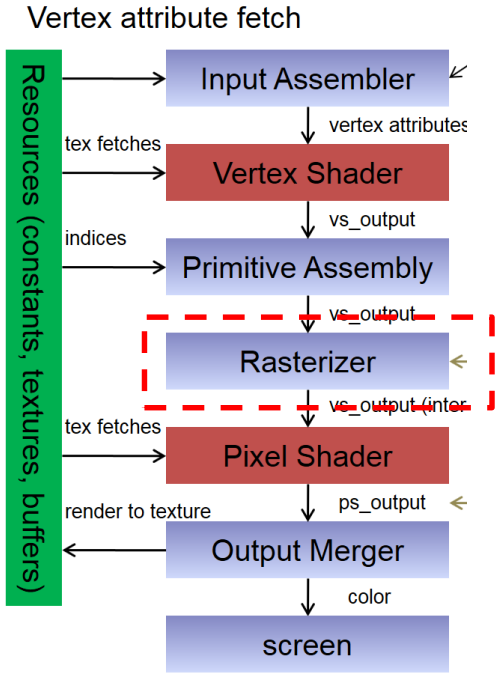
- В аппаратуре всё делается через треугольники

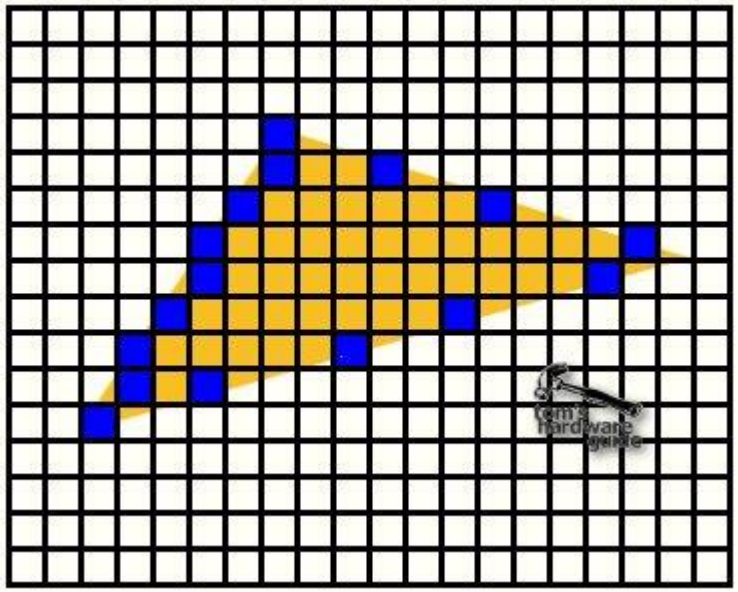


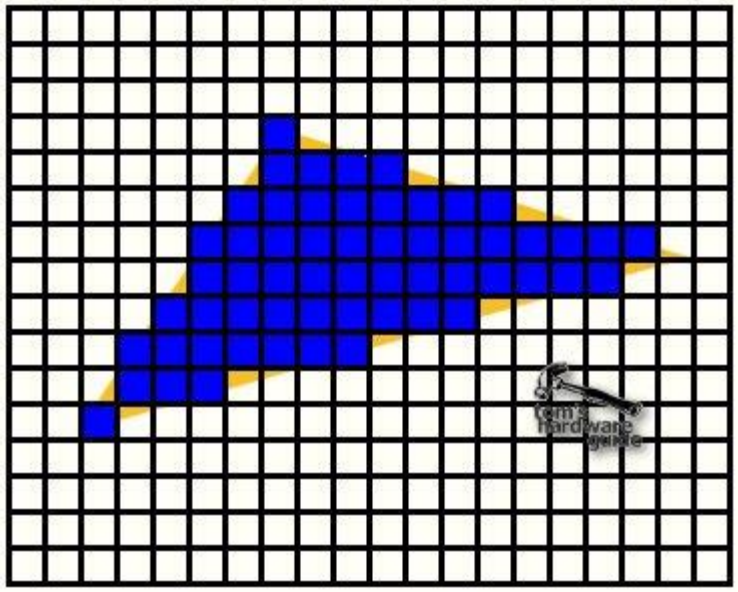


Растеризатор

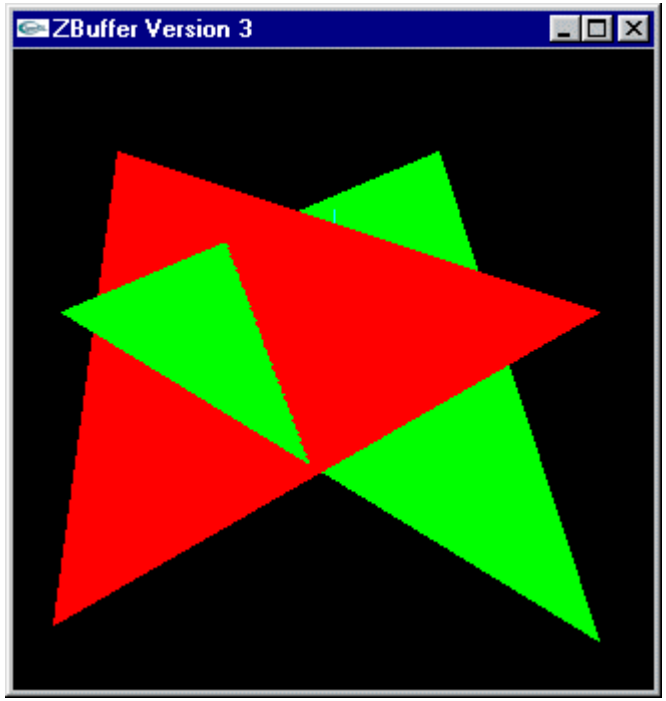
- Отсечение
- Растеризация
- Z-буфер (Early-z test)
- Буфер трафарета (Stencil test)
- Интерполяция атрибутов





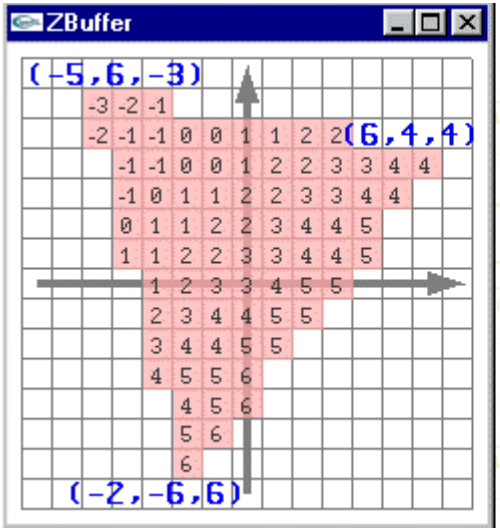


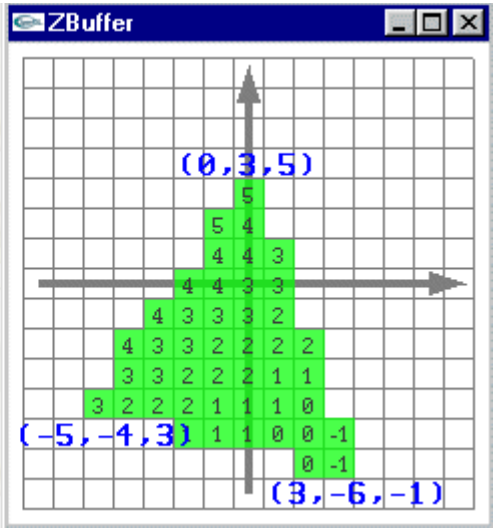
Растеризатор (Z-Buffer)

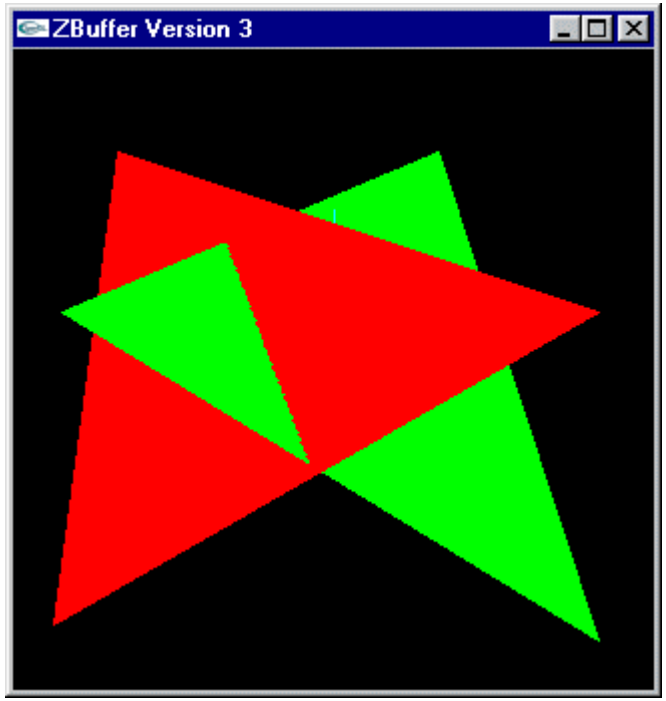


http://raphaello.univ-fcomte.fr/ig/zbuffer/zbuffer.htm

Растеризатор (Z-Buffer)

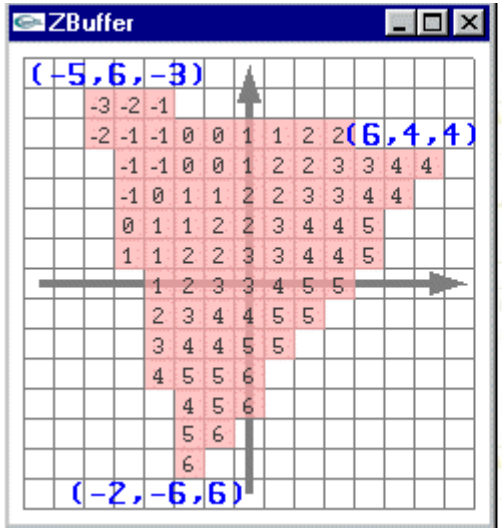


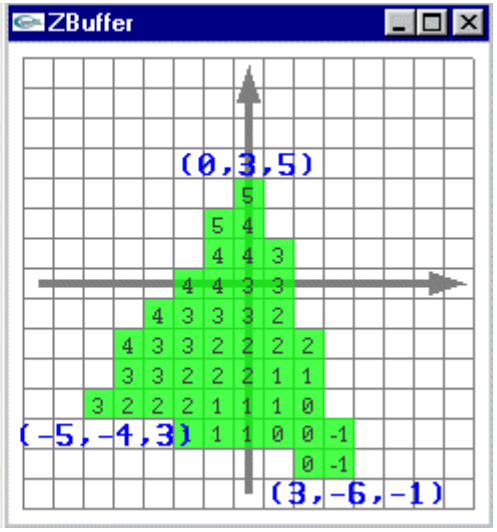


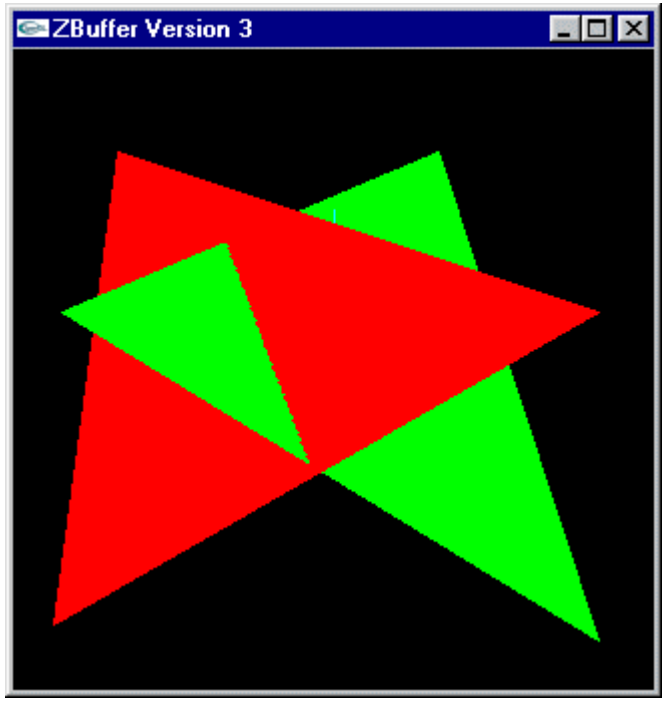


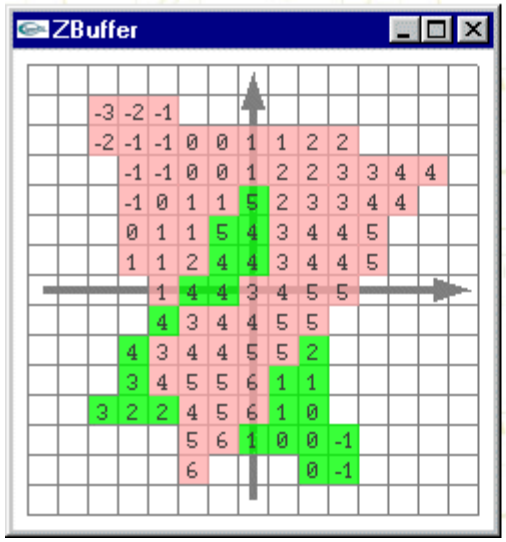
http://raphaello.univ-fcomte.fr/ig/zbuffer/zbuffer.htm

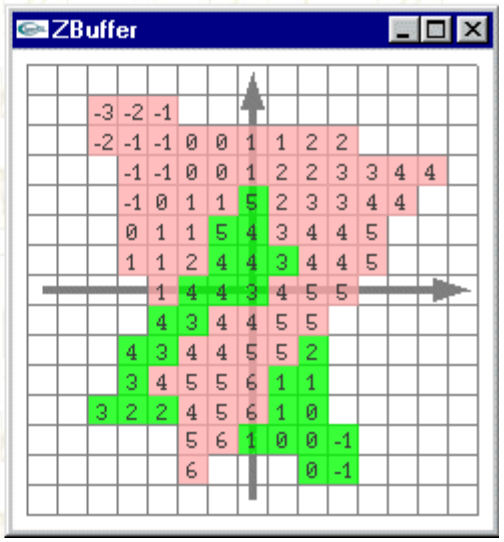
Растеризатор (Z-Buffer)









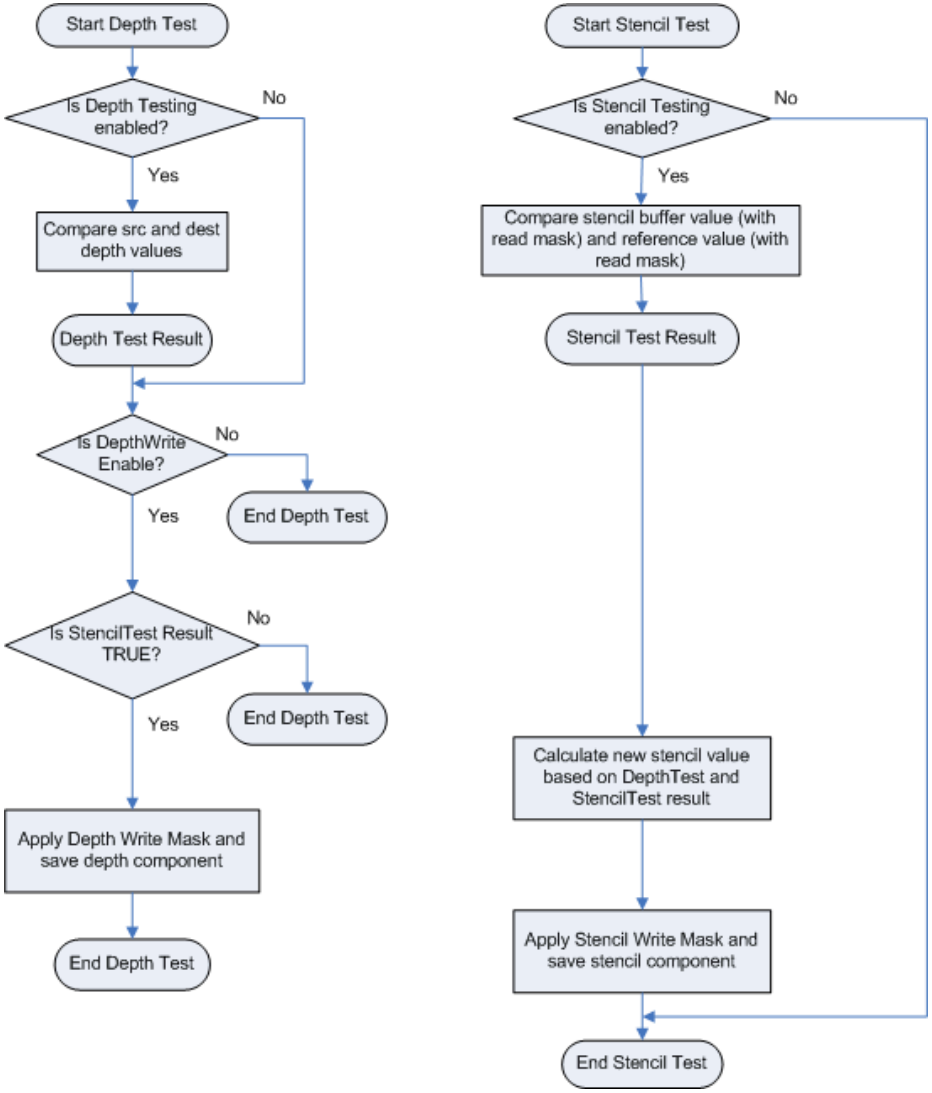


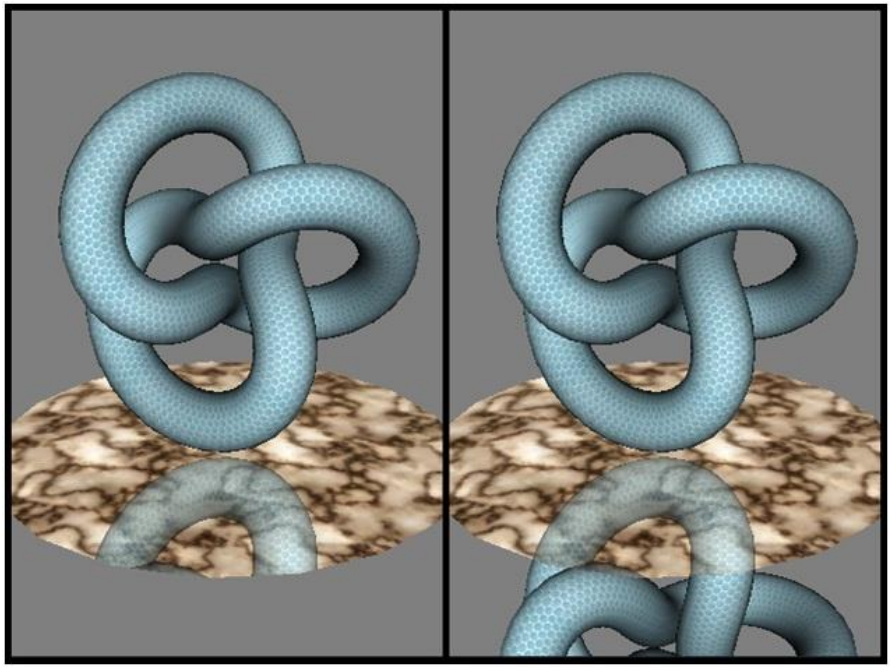
http://raphaello.univ-fcomte.fr/ig/zbuffer/zbuffer.htm

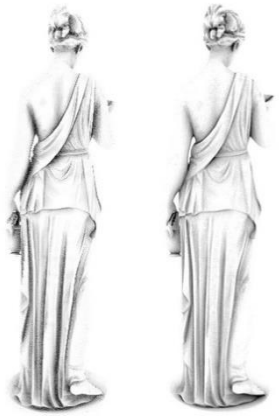
Z-buffer fighting

- 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);`

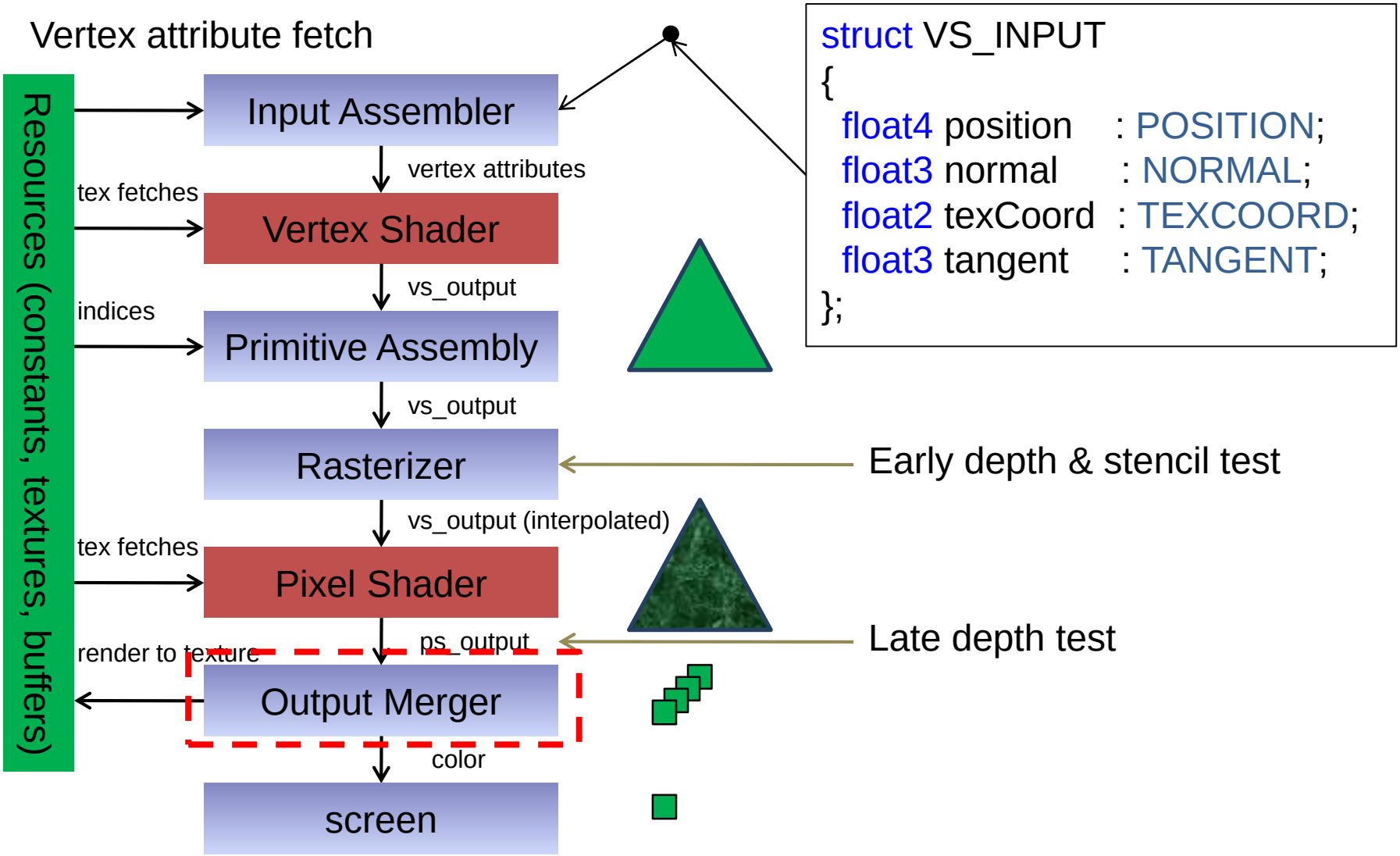
Depth & Stencil Test





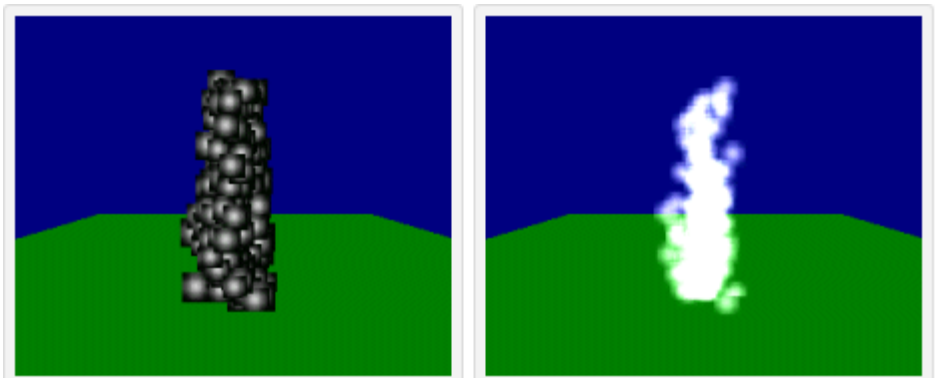


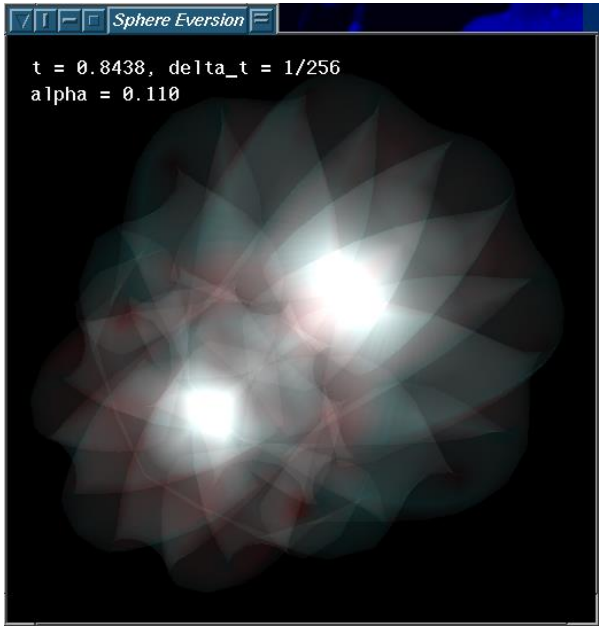
Графический конвейер и шейдеры



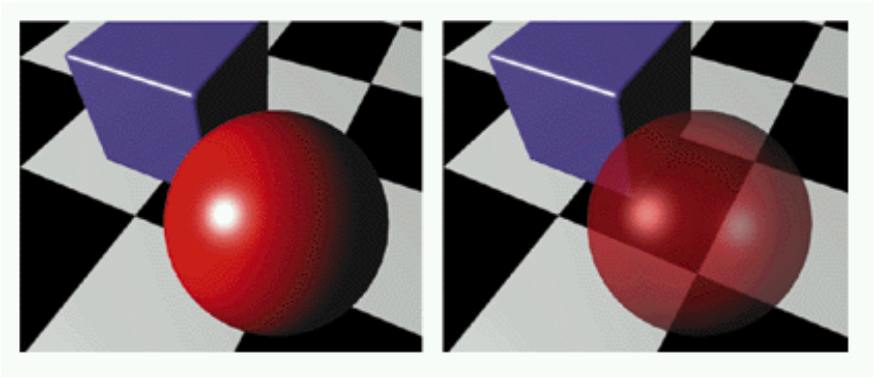
Output Merger

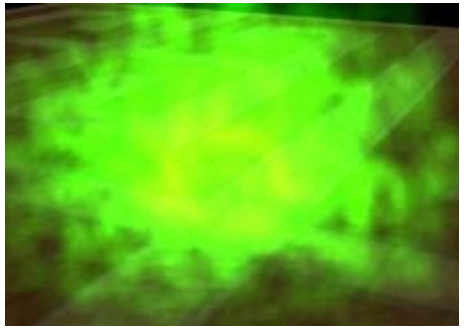
- Альфа-смешивание





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





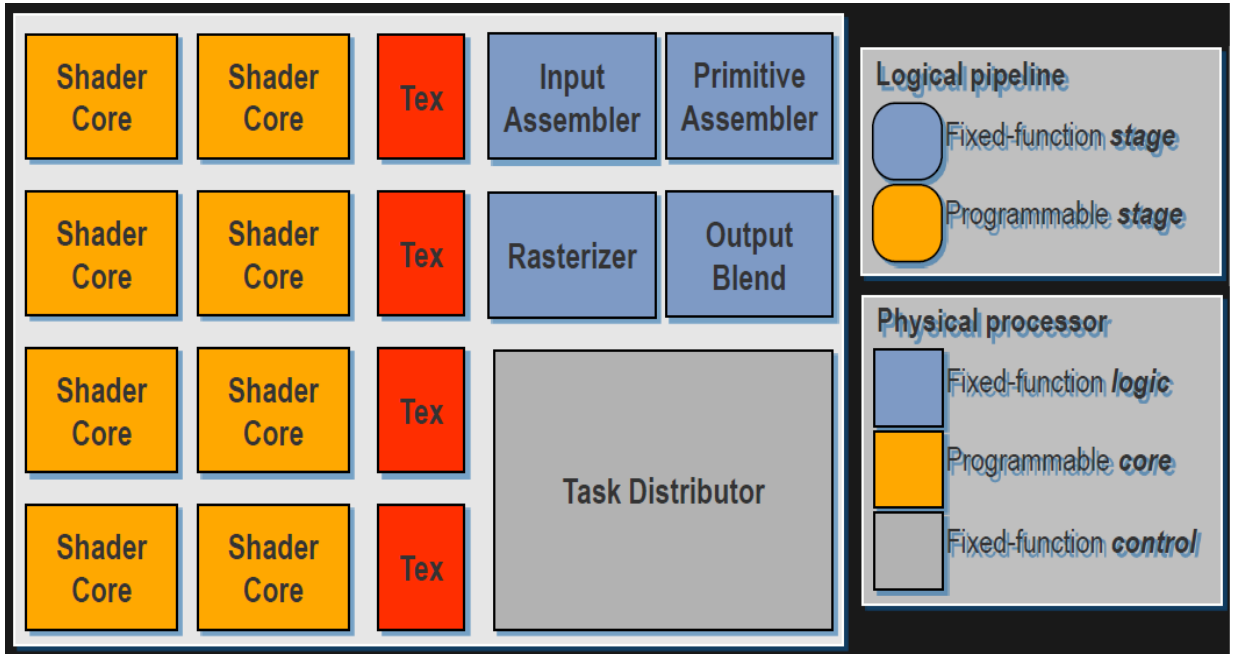
Output Merger

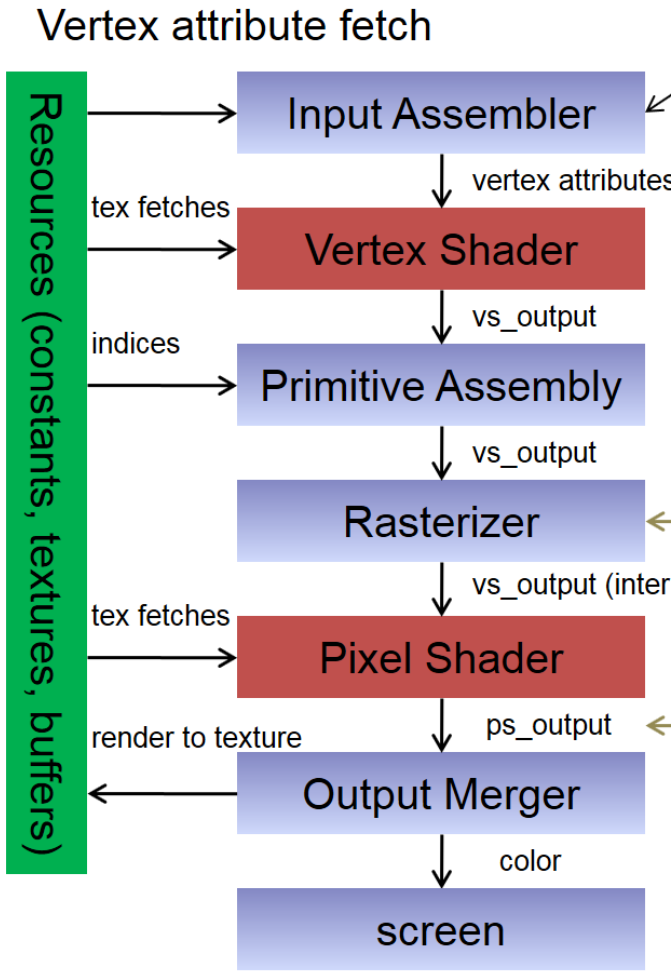
- Альфа-смешивание
  - glEnable(GL_BLEND)
  - glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
  - ...
  - glDisable();

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

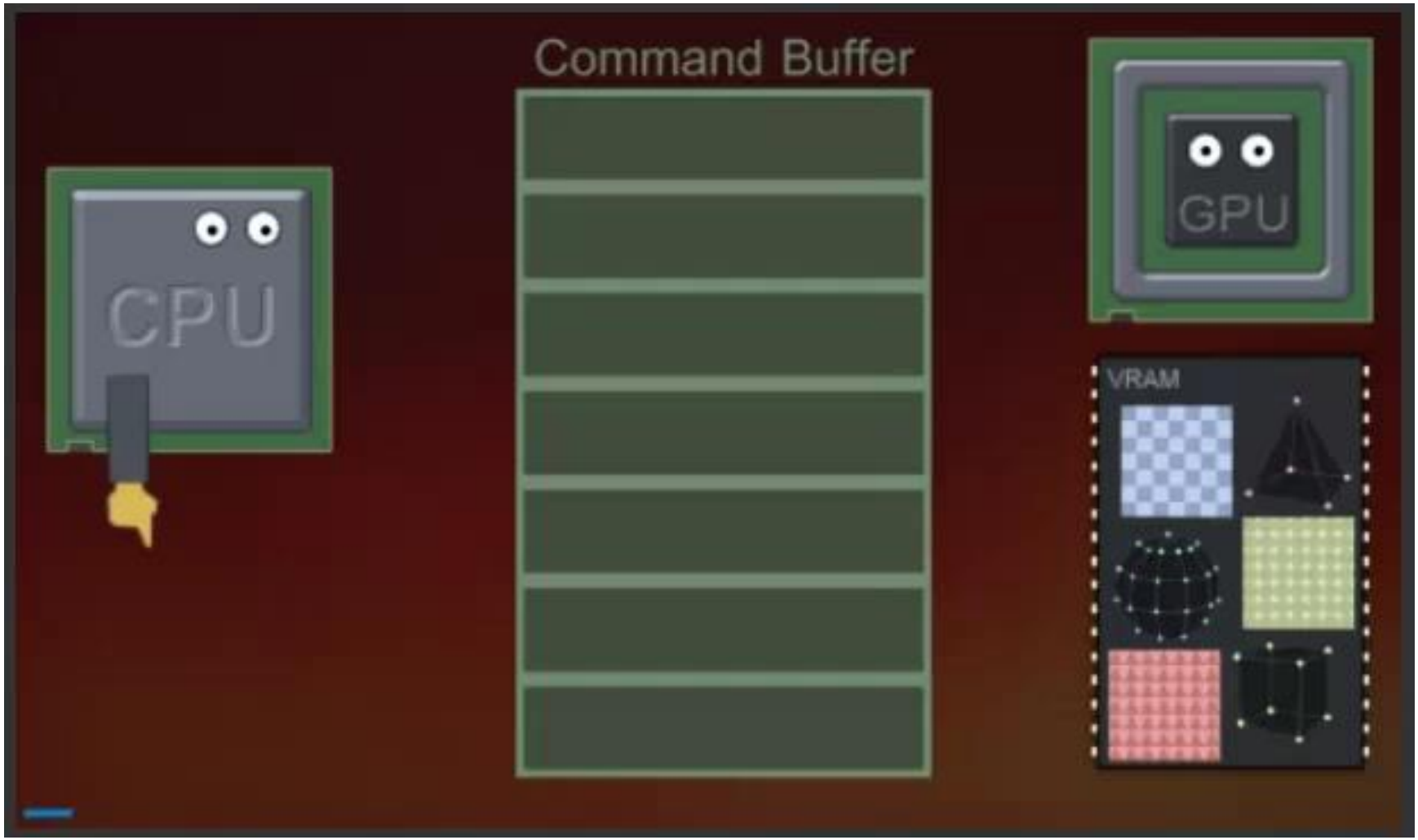
Графические процесоры

- Графика
- Вычисления общего назначения

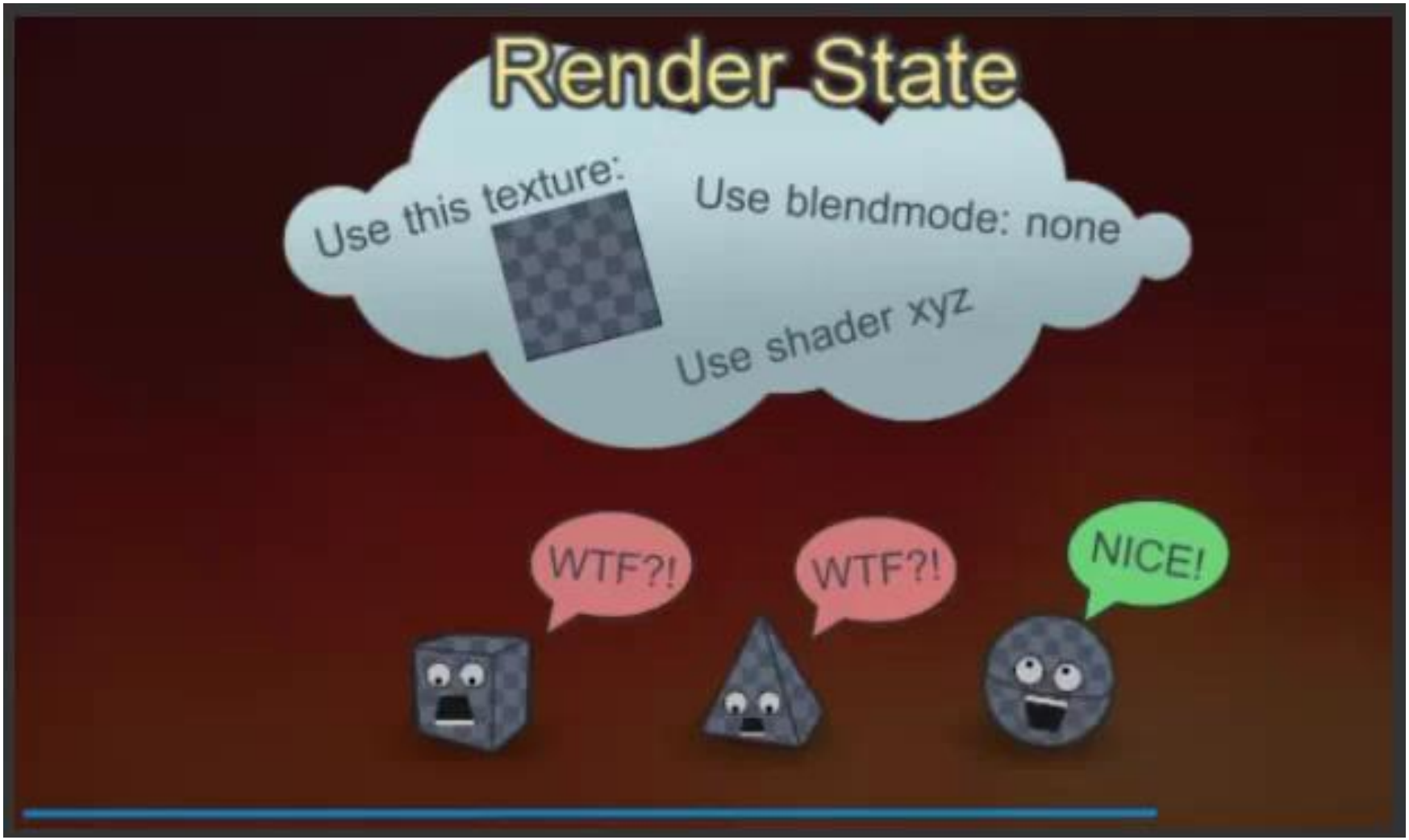




Графические процесоры



OpenGL


Render State
Use this texture:
Use blendmode: none
Use shader xyz
WTF?!
WTF?!
NICE!

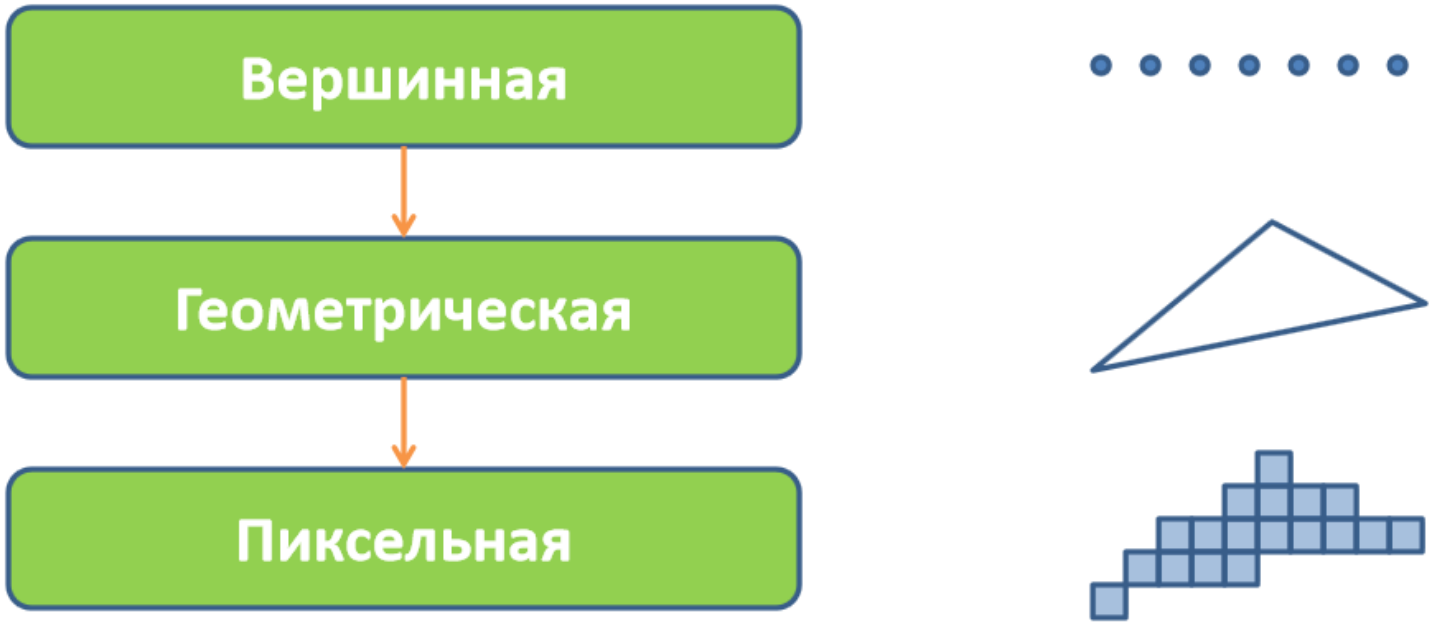
Графический конвейер (база)

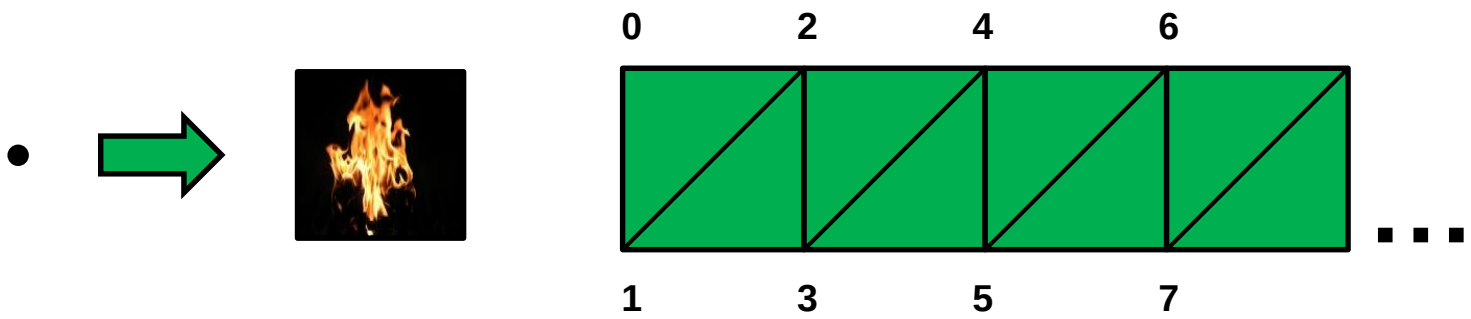
OpenGL 2.0 / DirectX9

- 2 основных программируемых стадии

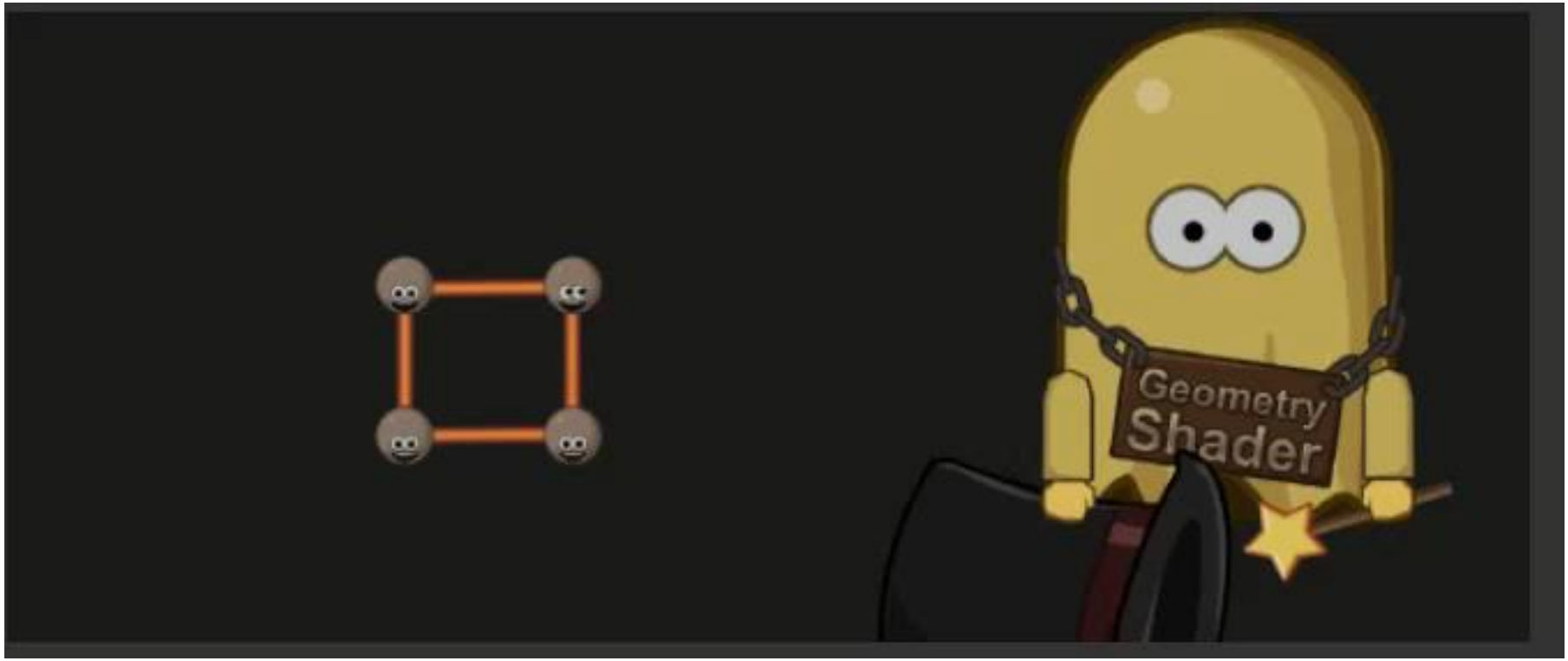
OpenGL 3.2 / DirectX 10

- 3 основных программируемых стадии



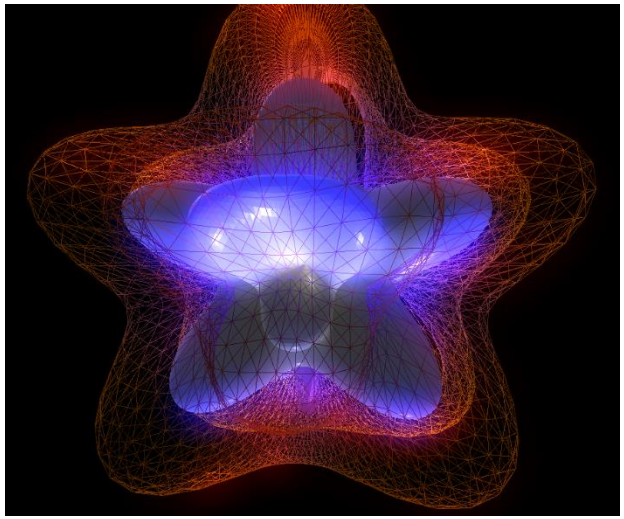


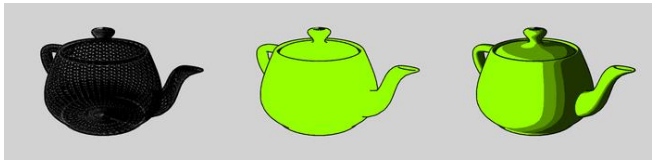
Геометрический шейдер

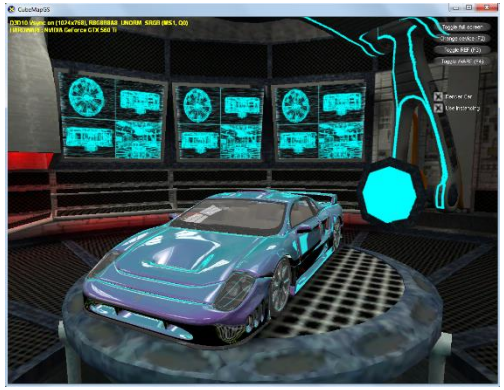


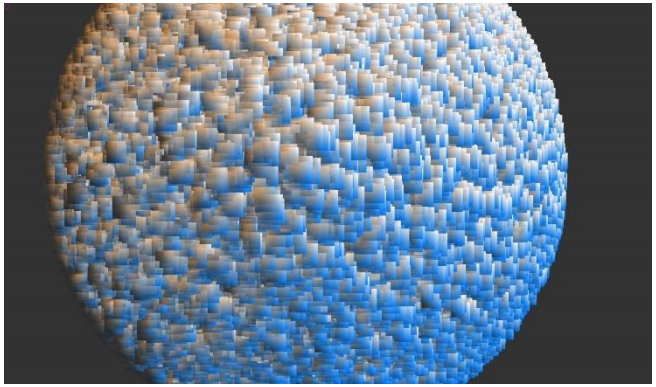
Геометрический шейдер

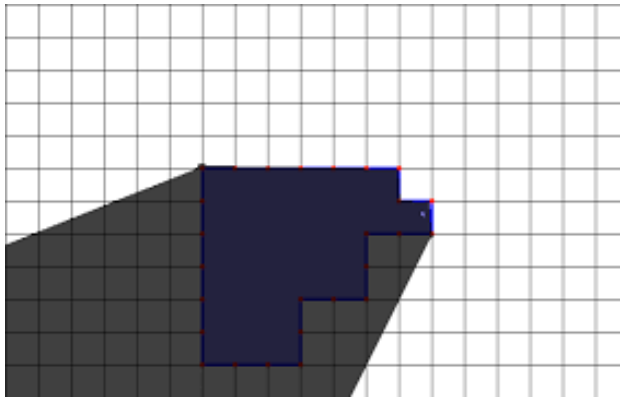
- Для чего нужен?
  - Рассчитать нормали
  - Генерация квадов (частицы)
  - Shadow Volume
  - CubeMapGS
  - Toone Shading



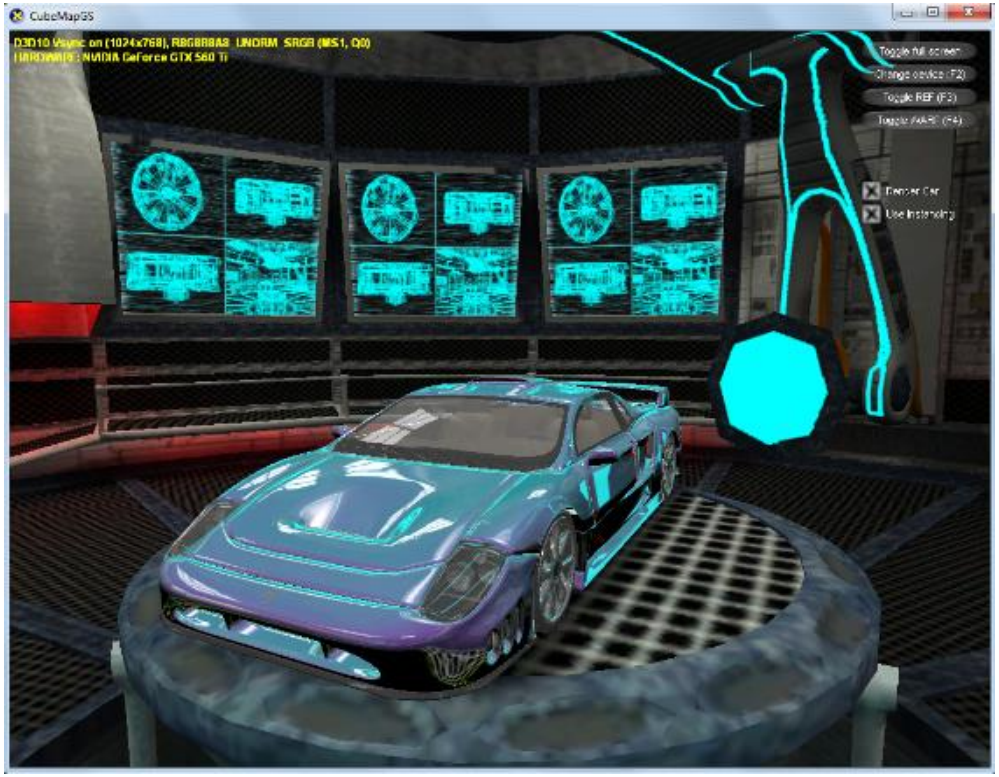


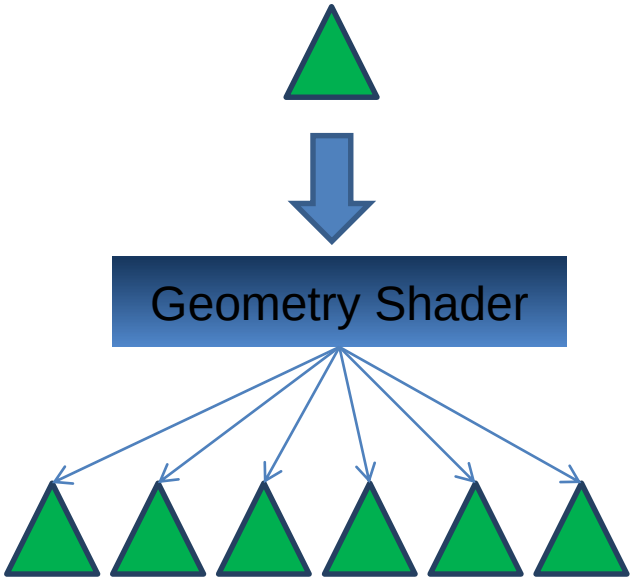


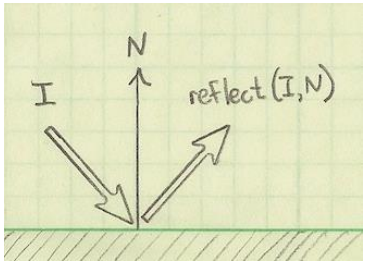


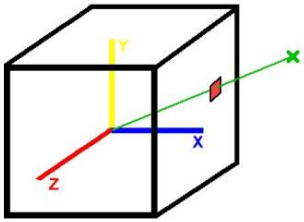


Геометрический шейдер

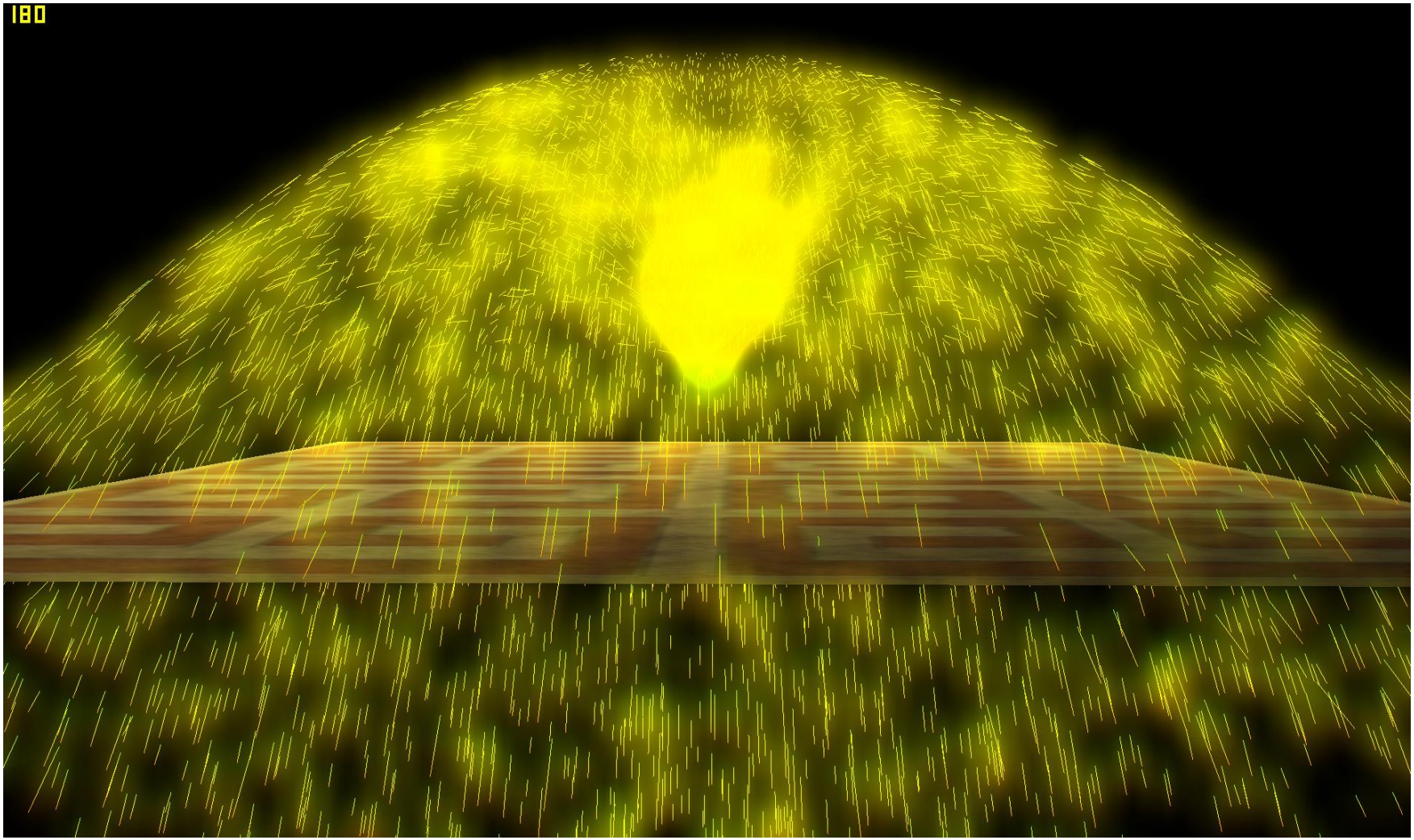








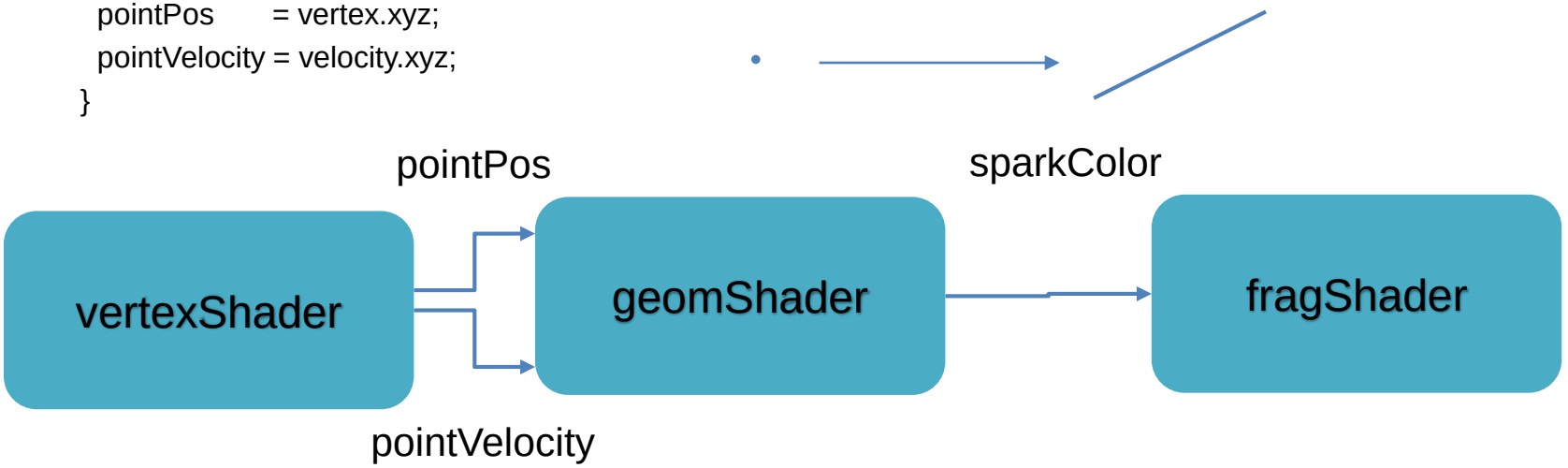
Геометрический шейдер



Вершинный и фрагментный шейдеры

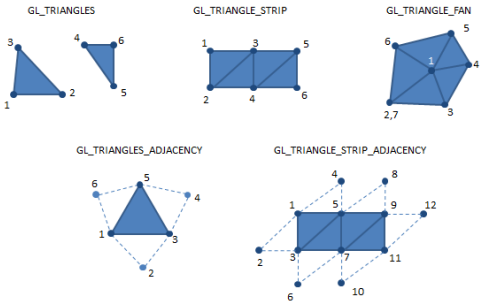
```
#version 400

in  vec4 vertex;
in  vec4 velocity;

out vec3 pointPos;
out vec3 pointVelocity;

void main(void)
{
  pointPos      = vertex.xyz;
  pointVelocity = velocity.xyz;
}
```

```
#version 400

in  vec3 sparkColor;
out vec4 fragColor;

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



Геометрический шейдер

```
#version 400
layout (points) in;
layout (line_strip, max_vertices = 2) out;

in  vec3 pointPos [];
in  vec3 pointVelocity [];

out vec2 fragmentTexCoord;
out vec3 sparkColor;

uniform mat4 projectionMatrix;
uniform mat4 modelViewMatrix;
uniform float delta_t = 0.1f;
```

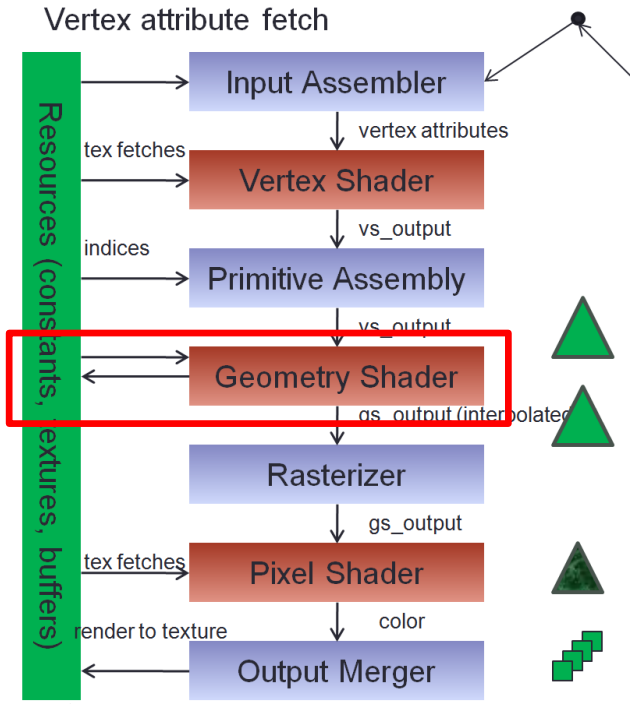



```
void main(void)
{
  vec3 pos = pointPos[0];
  vec3 vel  = pointVelocity[0];

  vec3 pos1 = pos - 0.75f*vel;

  gl_Position = projectionMatrix*(modelViewMatrix*vec4(pos,1));
  sparkColor  = 2*vec3(2,1,0);
  EmitVertex();

  gl_Position = projectionMatrix*(modelViewMatrix*vec4(pos1,1));
  sparkColor  = 2*vec3(1,2,0);
  EmitVertex();

  EndPrimitive();
}
```

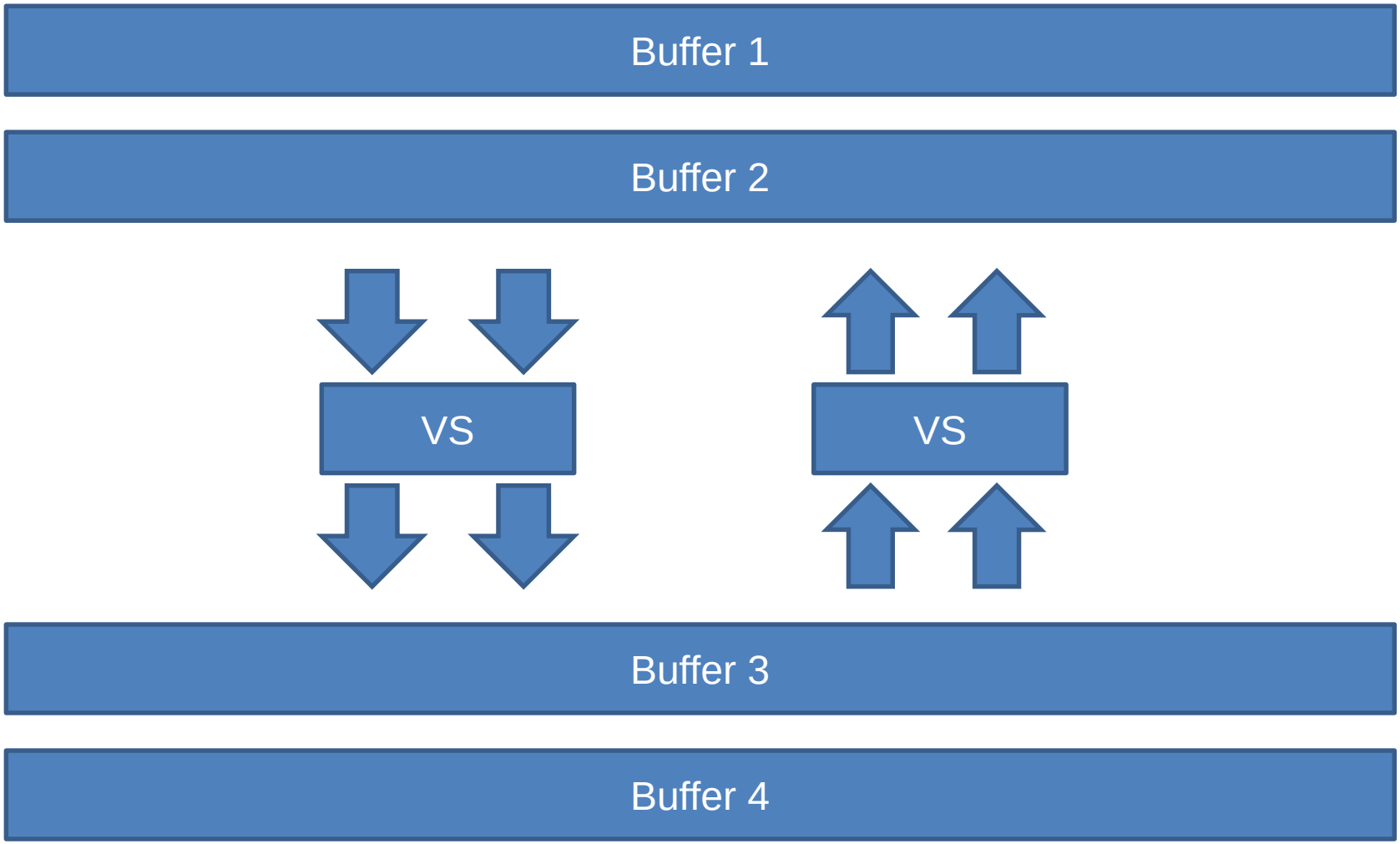
Transform Feedback, append-buffer

- 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

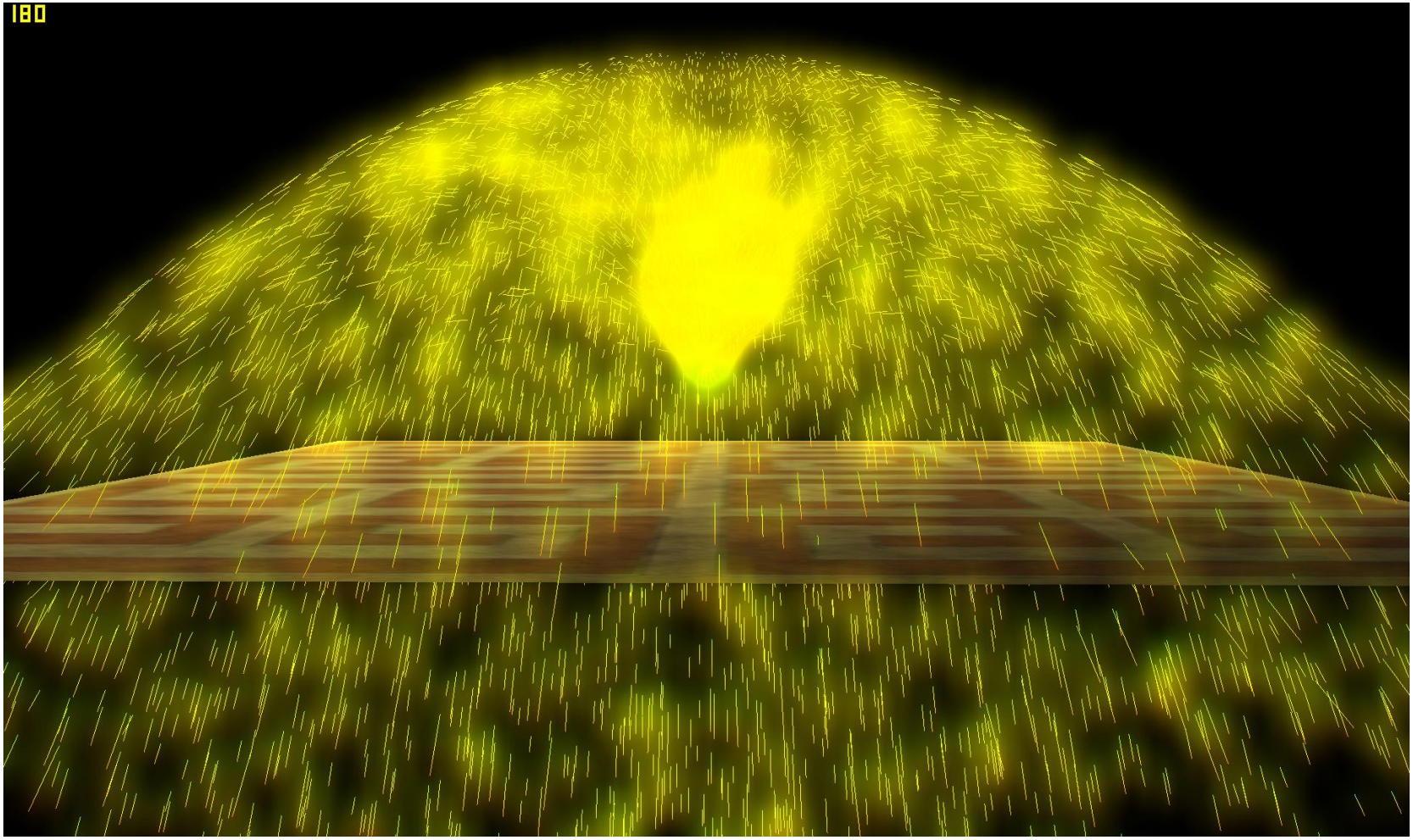


Stream Out & Transform feedback

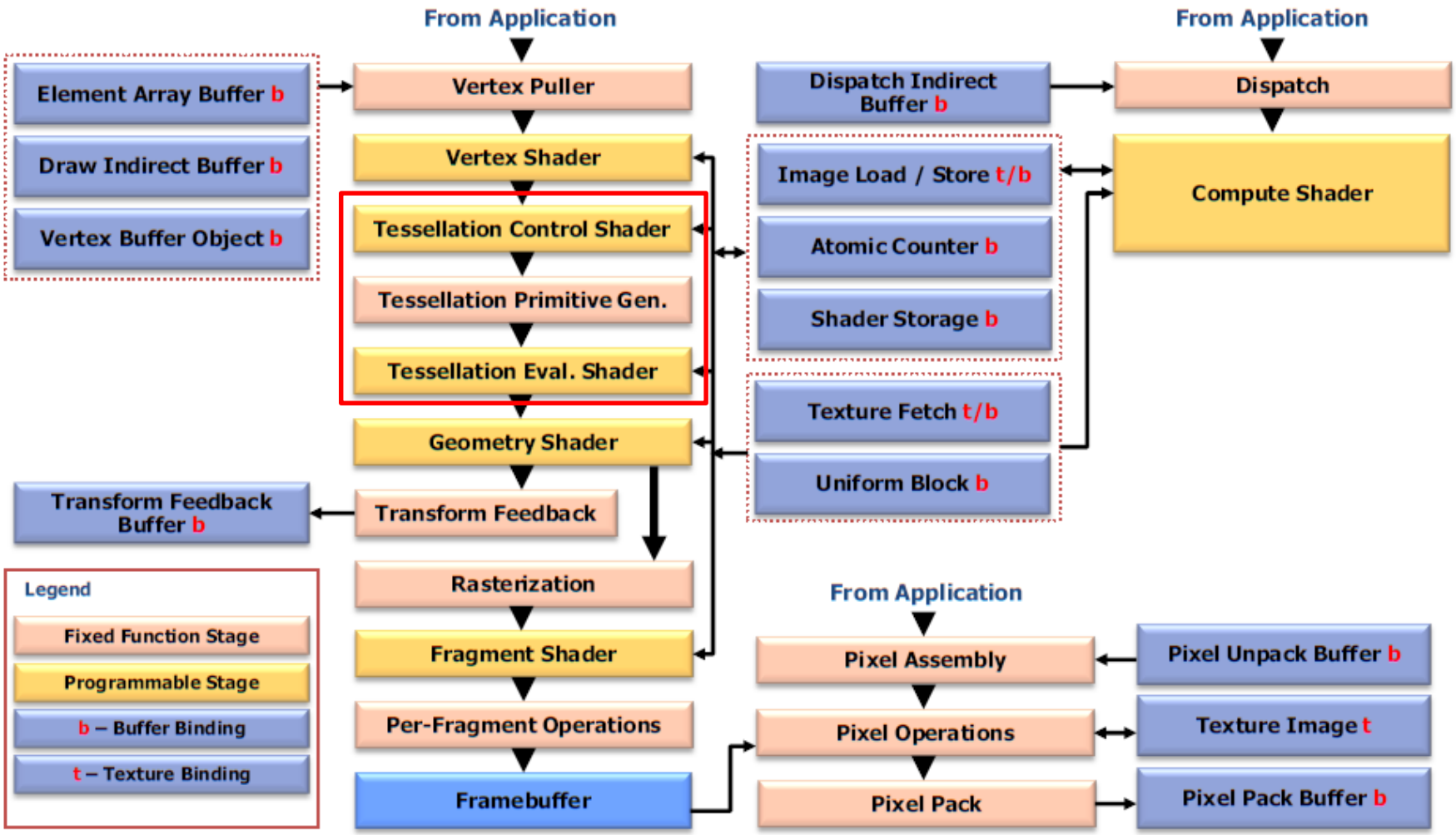


Stream Out & Transform feedback

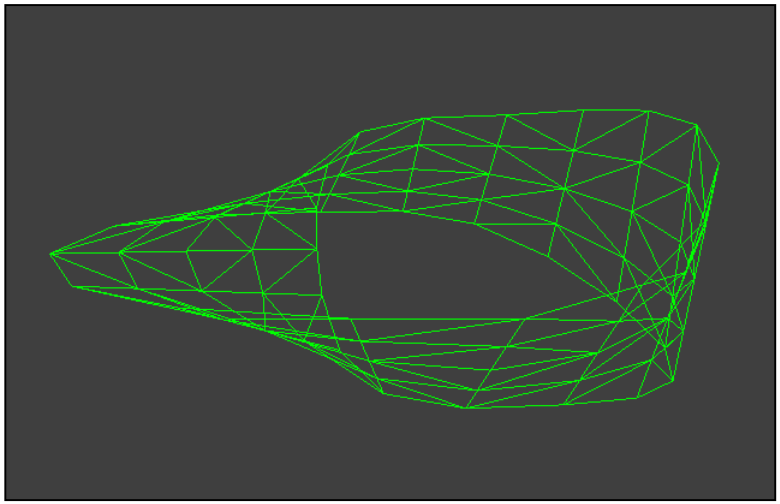
- Рассчет физики частиц

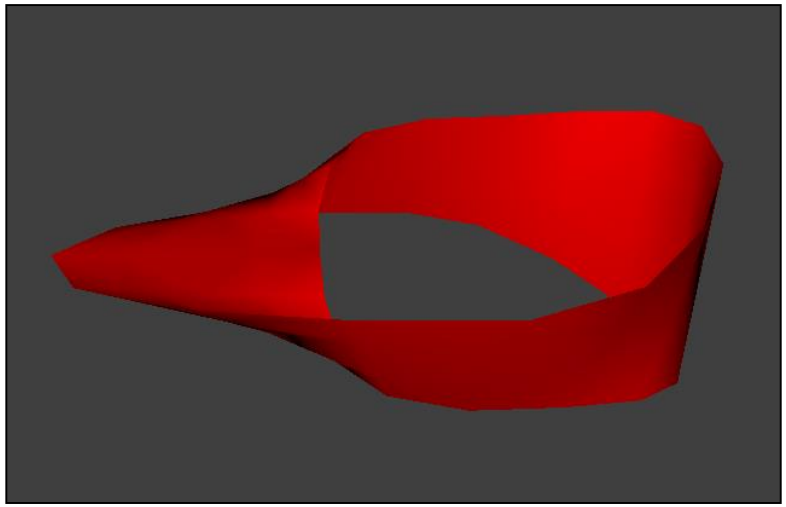


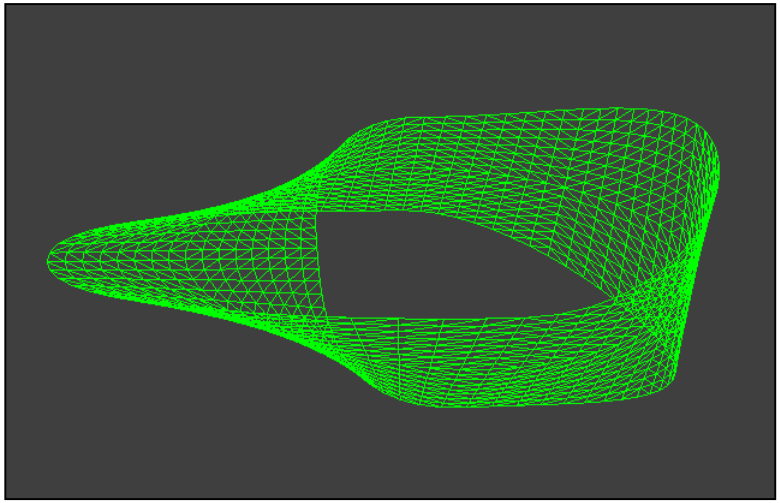
Тесселяция (кратко)

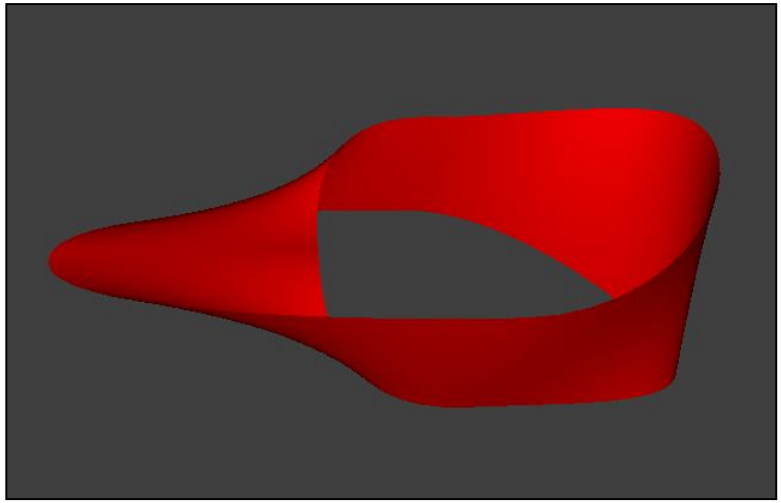


Тесселяция (кратко)

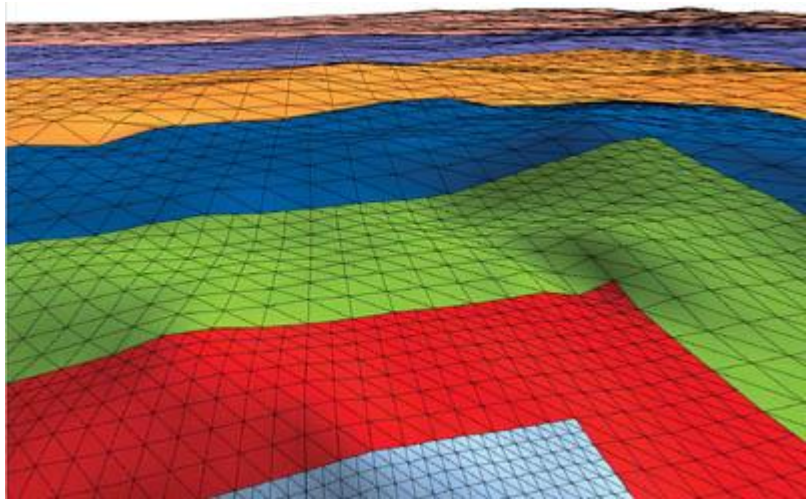


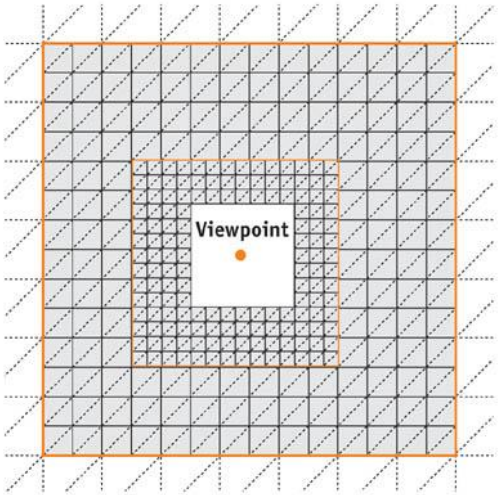






Тесселяция (кратко)





Графический конвейер

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(…);
```

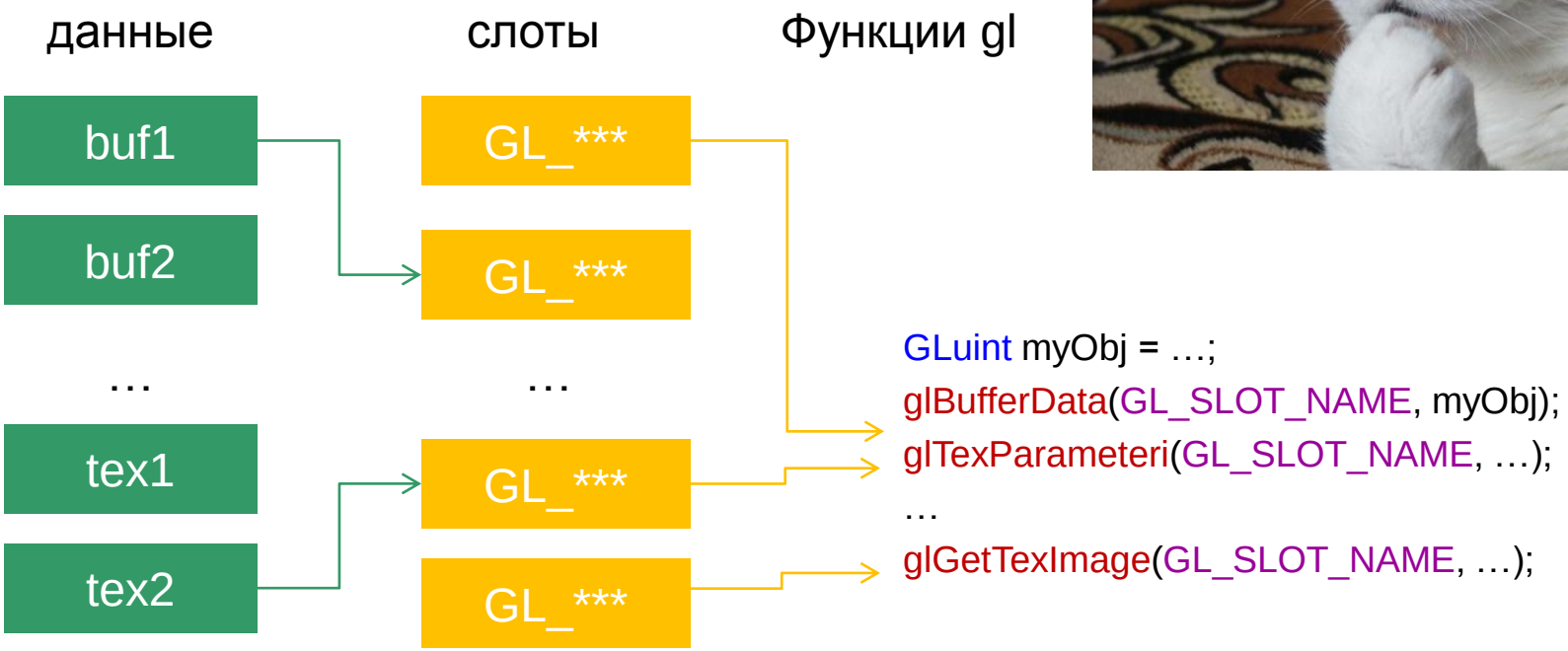
OpenGL

- Мульти-языковой ?
- 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

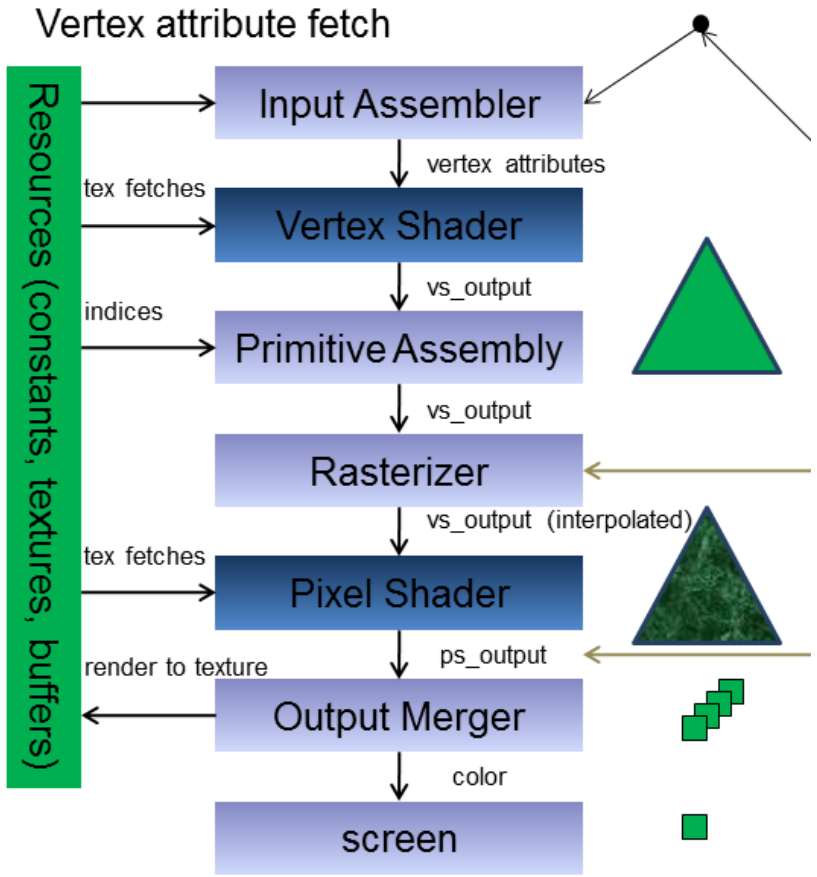
```
GLuint myObj = …;
glBindSomeObject(GL_SOME_TARGET_NAME, myObj);
glDoSomething(GL_SOME_TARGET_NAME, …);
```




```
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

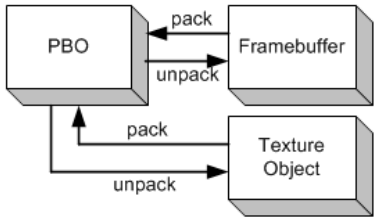
```
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, N*sizeof(GLfloat), (GLfloat*)cpuData, GL_STATIC_DRAW);
```

- 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

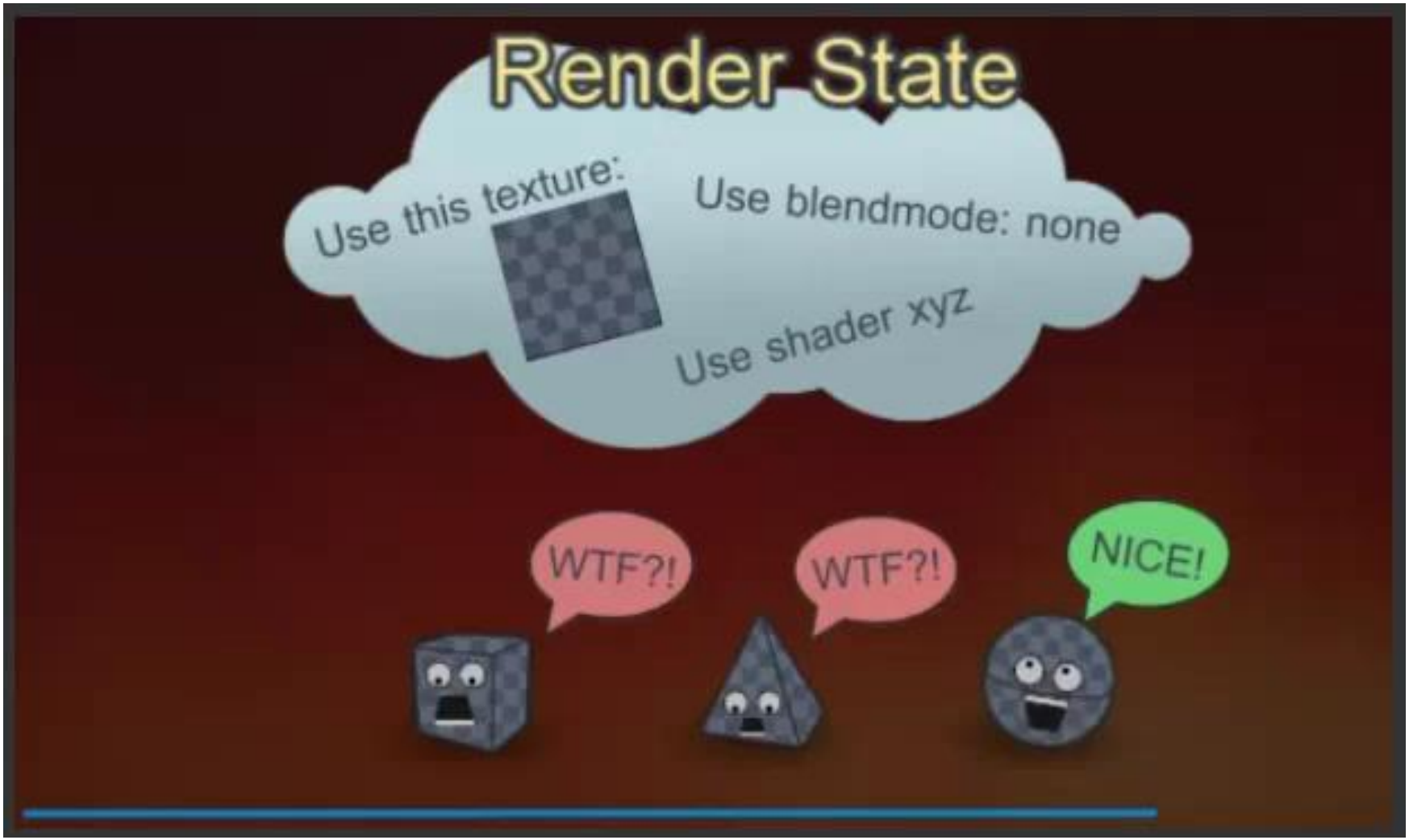
- GL, в шейдере:
  - Uniform переменные
  - UBO

```
// 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, Константы



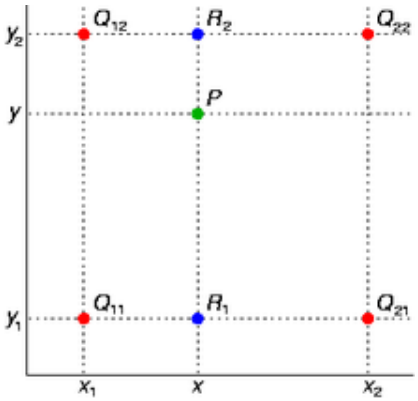
OpenGL, текстуры

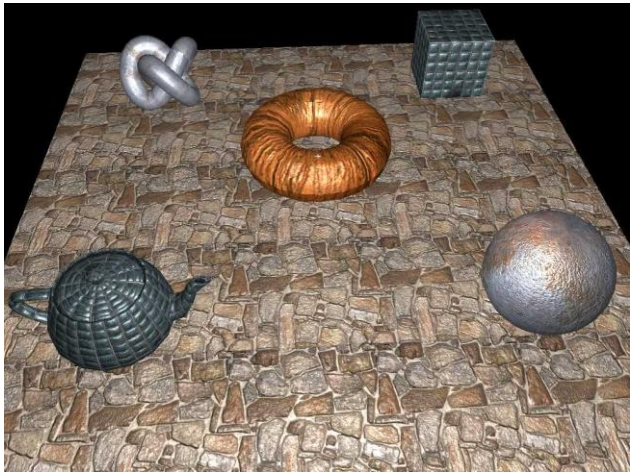
- Текстуры бывают:
  - GL_TEXTURE_1D
  - GL_TEXTURE_2D
  - GL_TEXTURE_3D
  - GL_TEXTURE_BUFFER
  - GL_TEXTURE_CUBE_MAP
  - GL_TEXTURE_1D_ARRAY
  - GL_TEXTURE_2D_ARRAY
  - GL_TEXTURE_CUBE_MAP_ARRAY
  - GL_TEXTURE_2D_MULTISAMPLE
  - GL_TEXTURE_2D_MULTISAMPLE_ARRAY

- 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

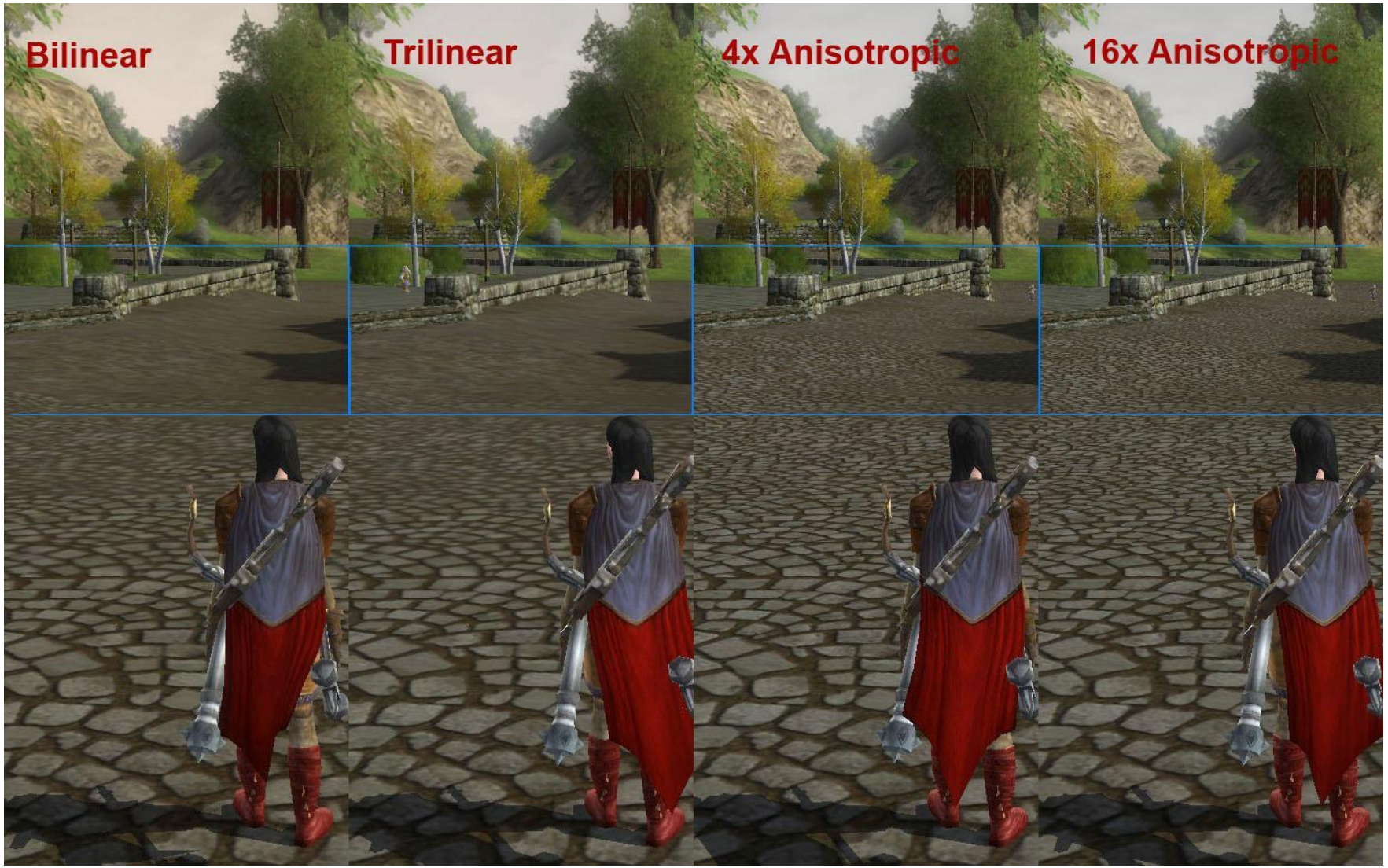
OpenGL, текстуры

- 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



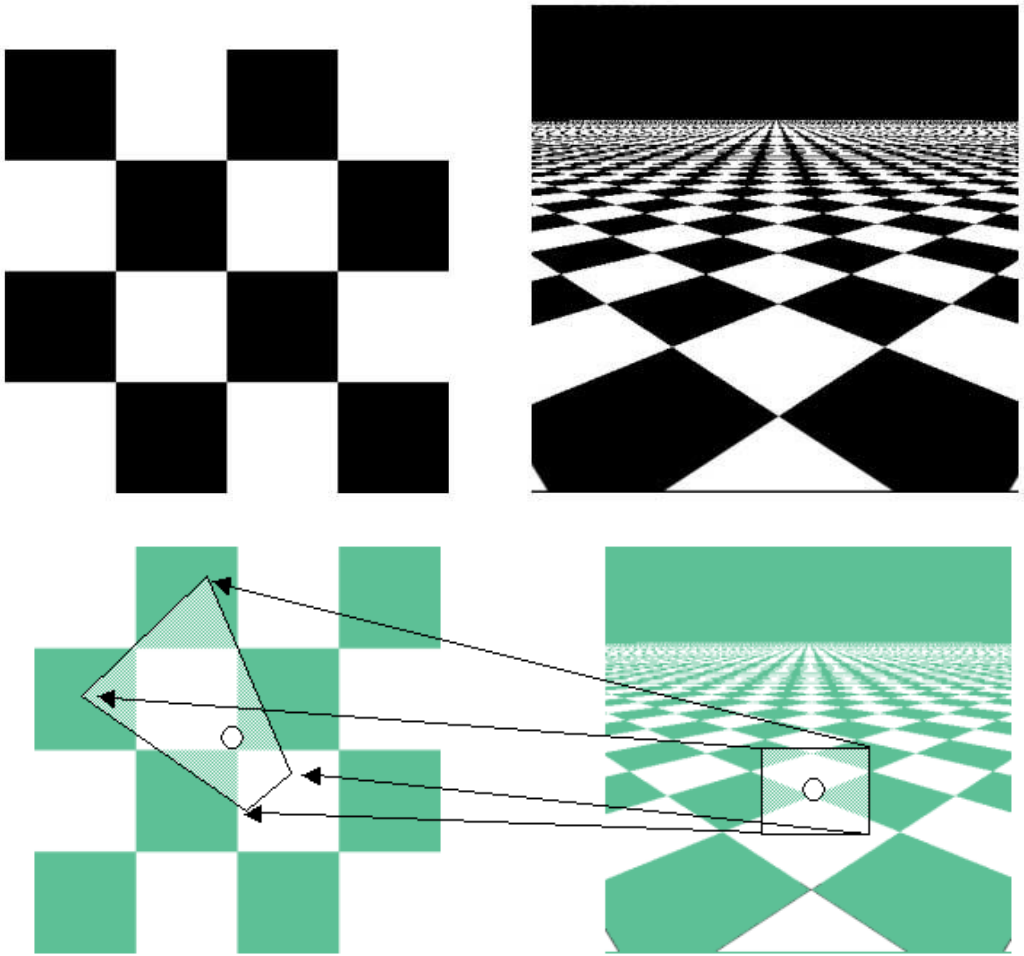


OpenGL, фильтрация текстур

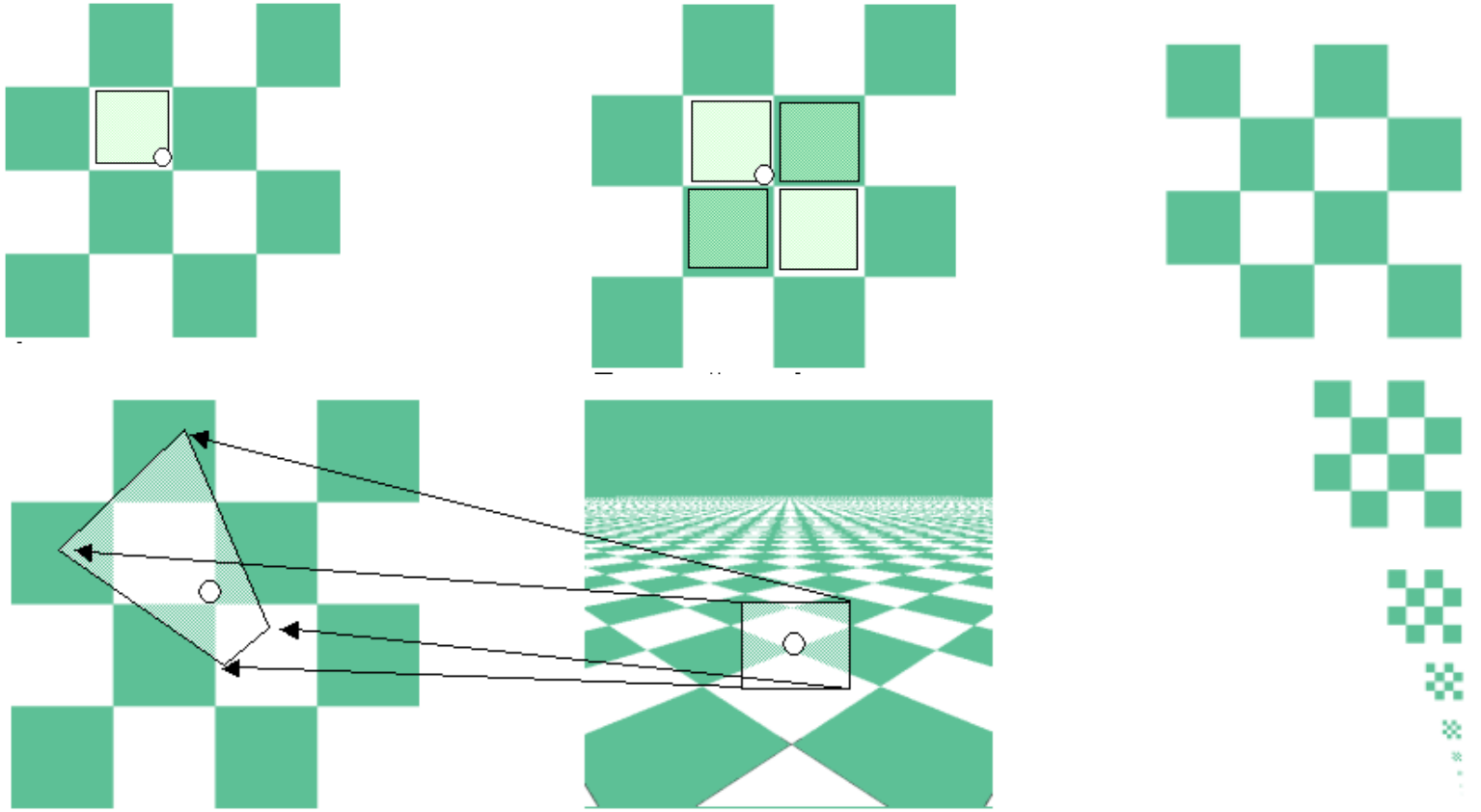


OpenGL, фильтрация текстур

- Почему нужна фильтрация?



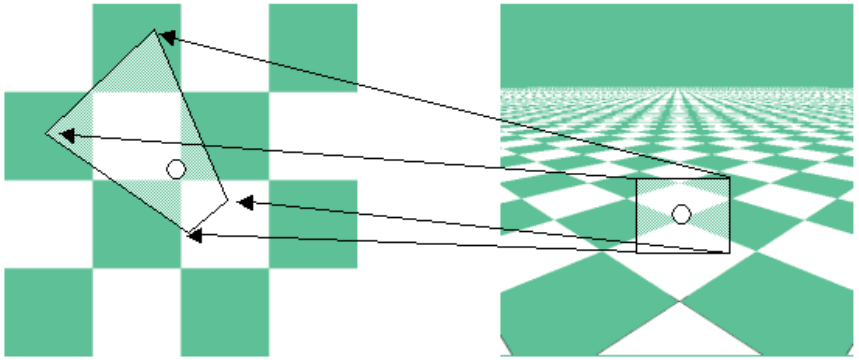
OpenGL, фильтрация текстур



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

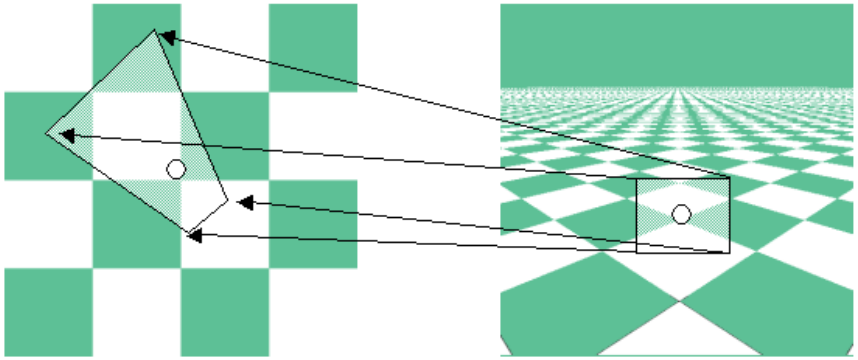
OpenGL, фильтрация текстур

- 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);
}
```

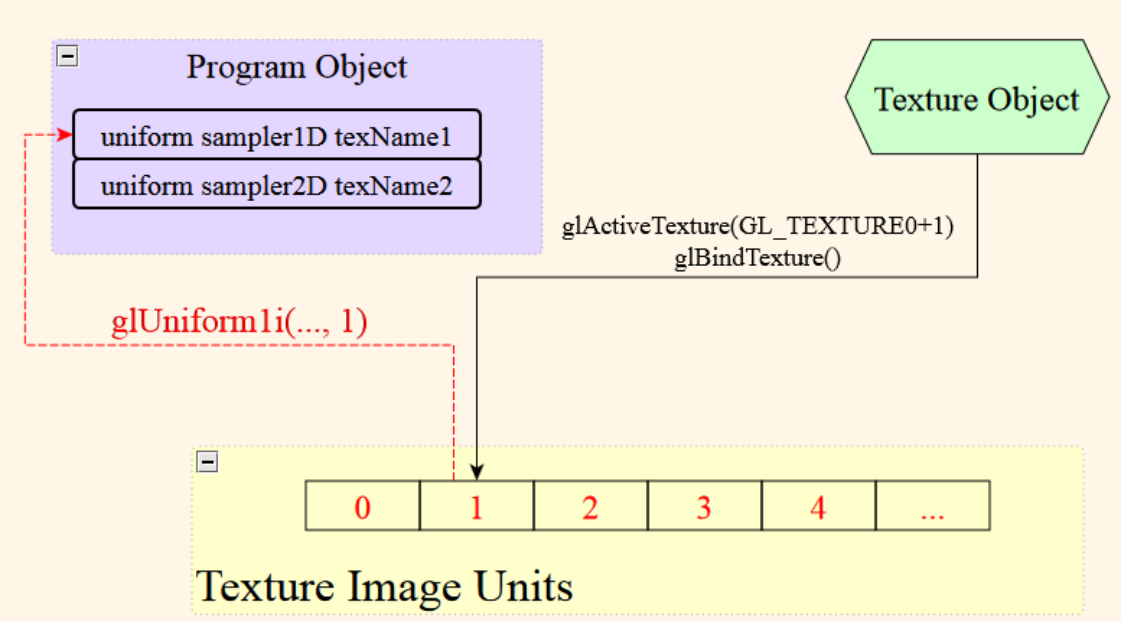
OpenGL, текстуры

- 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);
}
```



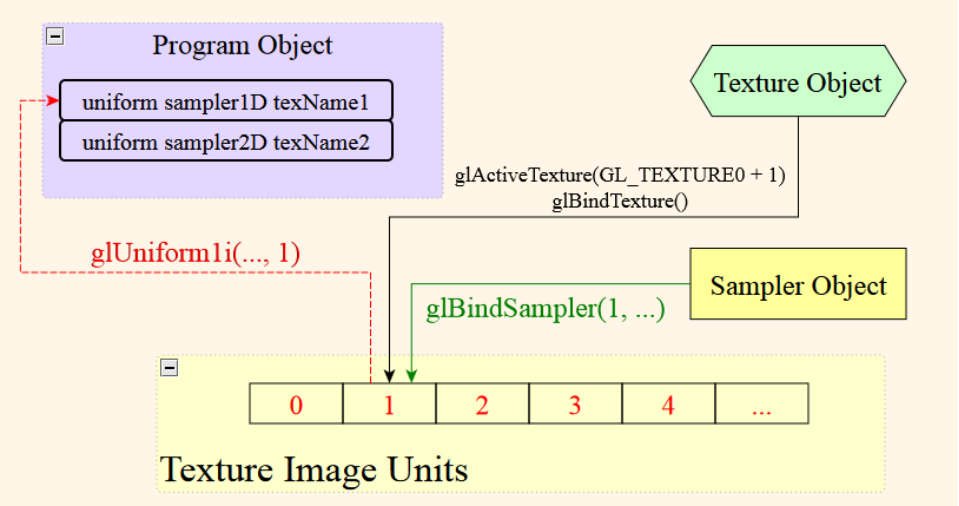
OGL3 glTexParameteri

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);
}
```



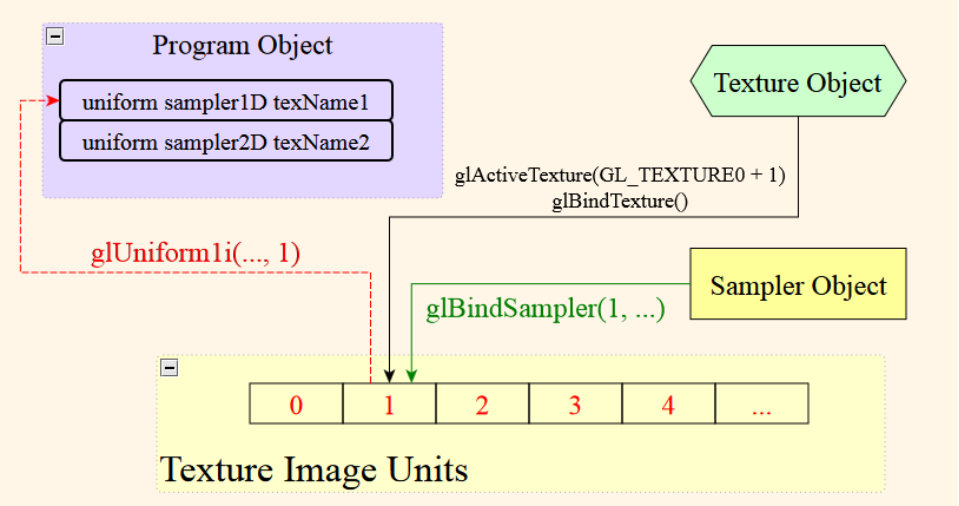
OGL4 Sampler Object

- 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);
}
```



Как работает GPU

Как работает GPU

Как работает GPU

OpenCL

clEnqueueWriteBuffer

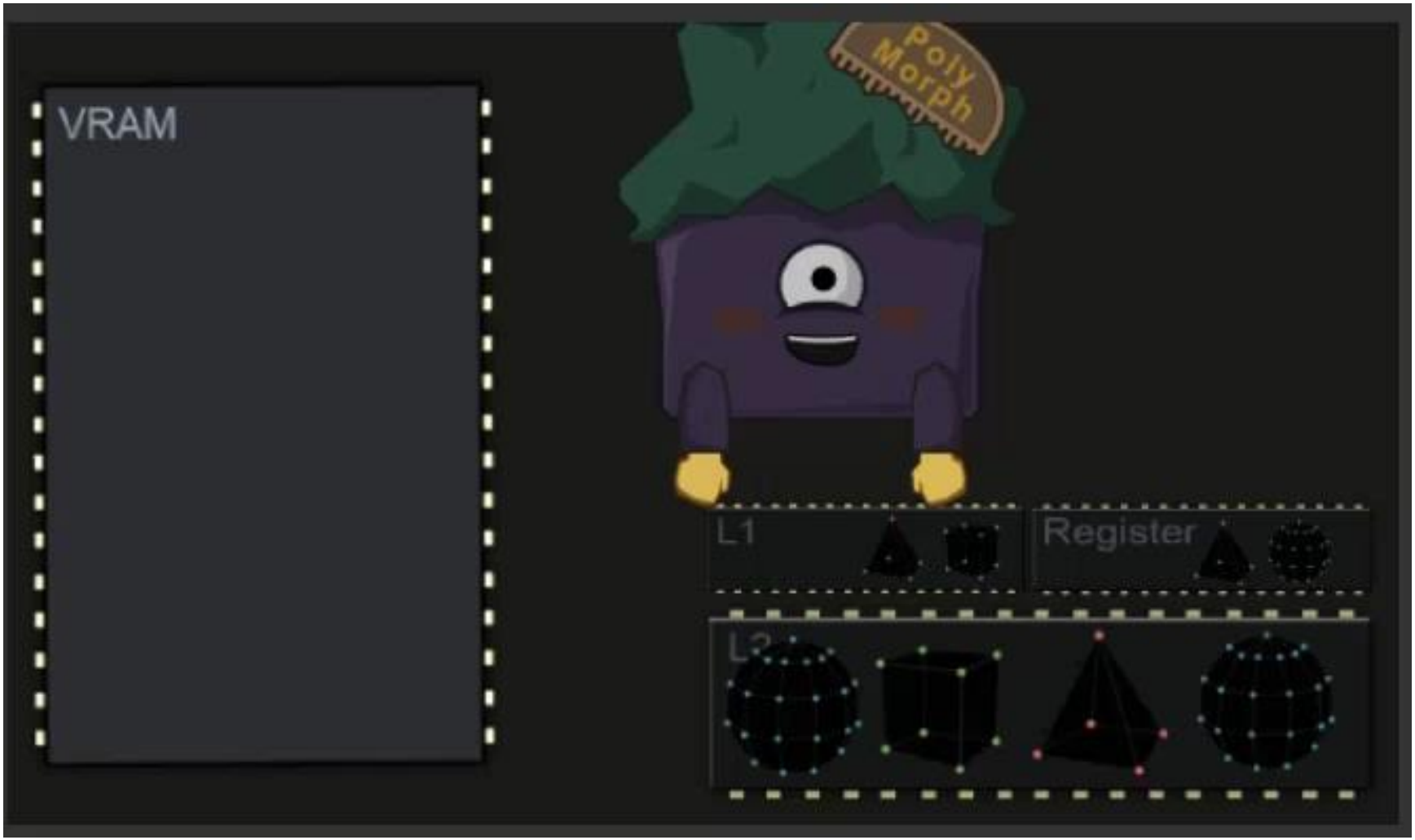
blocking_write

…

If blocking_write is CL_FALSE, the OpenCL implementation will use ptr to perform a **nonblocking** write. As the write is non-blocking the implementation can return immediately. **The memory pointed to by ptr cannot be reused by the application after the call returns**. The event argument returns an event object which can be used to query the execution status of the write command. When the write command has completed, the memory pointed to by ptr can then be reused by the application.

Как работает GPU

Как работает GPU

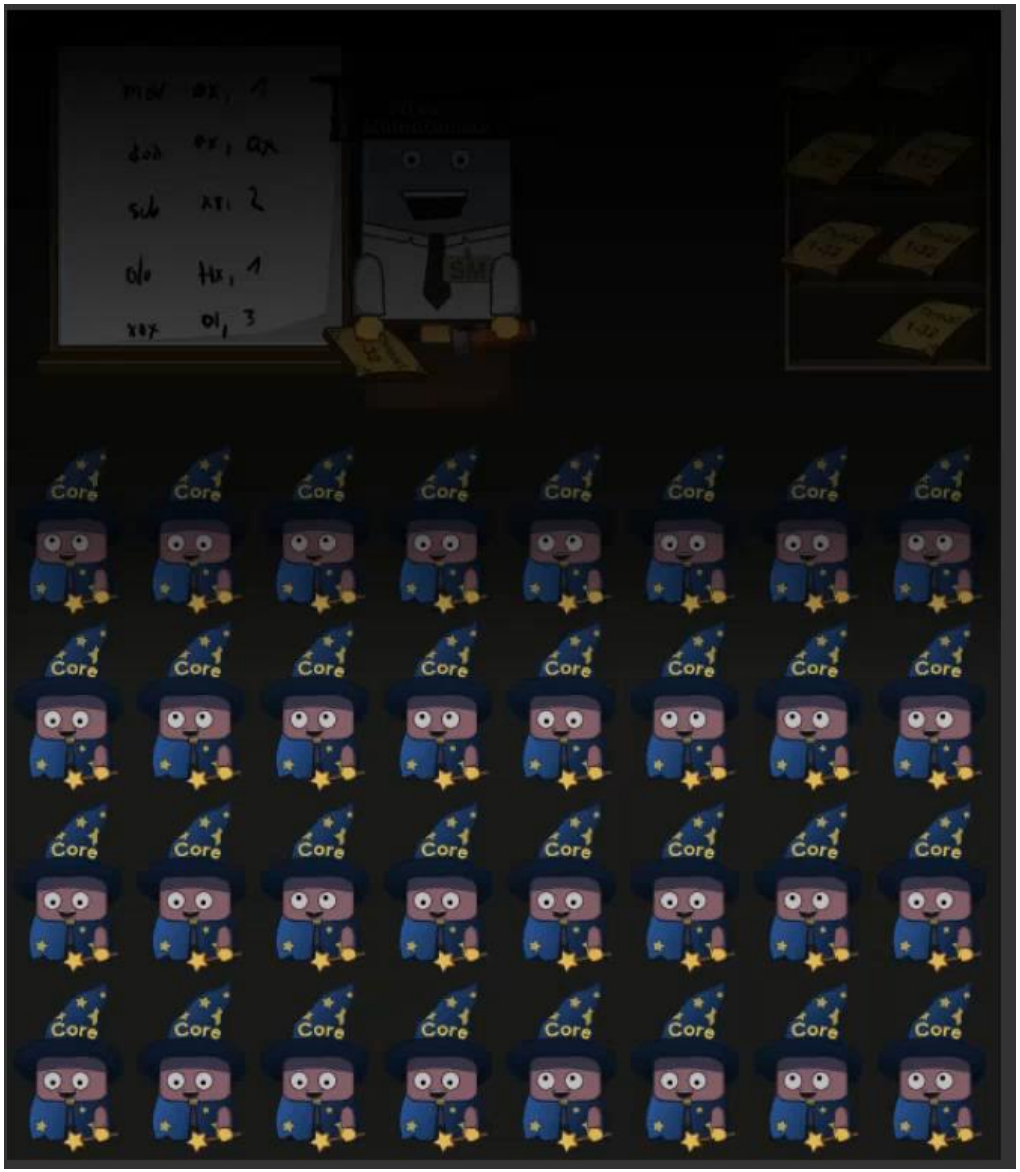


Особенности работы с GPU

Особенности работы с GPU

Особенности работы

Особенности работы GPU



Почему их много?

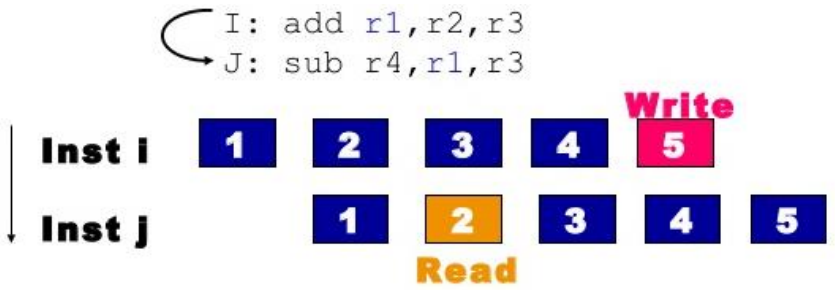
Почему их много?

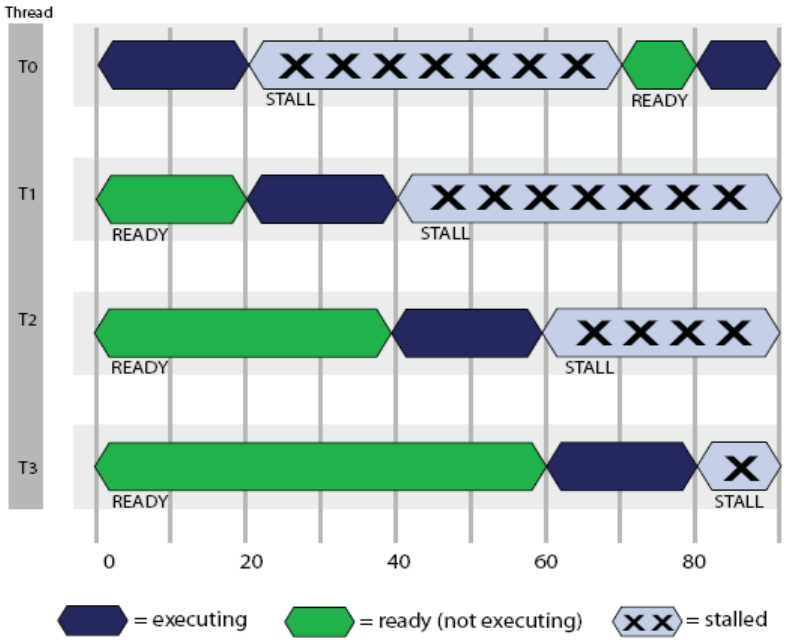
```
add R1,R2
mul R0,R1
```

```
add R1,R2 *
mul R0,R1 *
```

```
add R1,R2 **
mul R0,R1 **
```

```
add R1,R2
add R1,R2 *
add R1,R2 **

mul R0,R1
mul R0,R1 *
mul R0,R1 **
```





Отложенное затенение

- Deferred shading