

OpenGL

- a quick guide

Ulf Assarsson
Department of Computer Engineering
Chalmers University of Technology

Labs (= Tutorials)

- Some tutorials are on concepts treated on lectures at a later time.
 - When studying theory, it is beneficial to have some practice first...

And

- When doing tutorials, it is beneficial to have some theory first...
- Tradeoff
 - For practical reasons, we cannot have all theory in advance, so you get a bit of both worlds. The most important theory is often covered by lectures first.

Course strategy

- This course is more theory focused
 - Hardware acceleration evolves
 - Thus, implementation details change over time, while algorithms mostly stay the same.
 - Better to learn the algorithms, and look up hardware functionality at time of implementation
- Overview course
 - Less focus on details, which you can lookup yourself when you need them and if you are aware of the main concept.
- There will be half-time wrapup slides and full-time repetition slides
 - Covering **all** important topics for you on this course.

Graphics APIs

- OpenGL
- DirectX
- Vulkan
- Metal
- WebGPU

OpenGL vs Direct3D

- Direct3D

- Microsoft, Sept. '95 on Windows95
- Common for games
- Historically: “Adapted to graphics hardware evolution”
 - Now: influences hardware features perhaps more than OpenGL

**Direct3D was
messy to program
version 3.0 – 6.0.**

Today version 12

- OpenGL

- From SGI GL 1982
- Historically:
 - “Precede the hardware evolution”
 - Operation system independent
 - Window system independent
 - Industry, games (Quake –John Carmack, Apple, Linux)
 - January 1992
 - Extendable, stable, designed for how you *want* to program graphics,

**Today
version 4.6**

Compatibility...

E.g., getting OpenGL 4.1

```
SDL_GL_LoadLibrary(nullptr); // Default OpenGL is fine.
```

```
// Request an OpenGL 4.1 context (should be Core for us)
```

```
// - Most Macs support 4.1 and requires Core 4.1 exactly
```

```
// - Some Intel graphics cards only support Core
```

```
// - NVIDIA's nsight (profiling tool) requires Core.
```

```
SDL_GL_SetAttribute(SDL_GL_ACCELERATED_VISUAL, 1);
```

```
SDL_GL_SetAttribute(SDL_GL_CONTEXT_PROFILE_MASK, SDL_GL_CONTEXT_PROFILE_CORE);
```

```
SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION, 4);
```

```
SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION, 1);
```

```
SDL_GL_SetAttribute(SDL_GL_CONTEXT_FLAGS, SDL_GL_CONTEXT_DEBUG_FLAG);
```

```
// Also request a depth buffer
```

```
SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
```

```
SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE, 24);
```

```
// Create the window
```

```
SDL_Window* window = SDL_CreateWindow(caption.c_str(), SDL_WINDOWPOS_UNDEFINED,
```

```
SDL_WINDOWPOS_UNDEFINED, width, height, SDL_WINDOW_OPENGL | SDL_WINDOW_RESIZABLE);
```

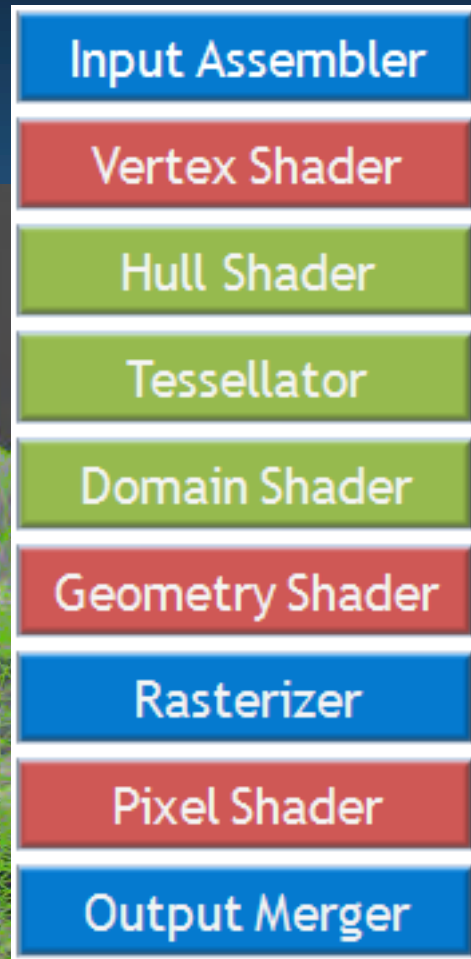
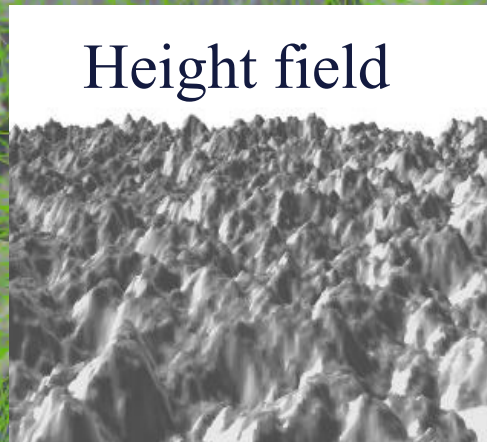
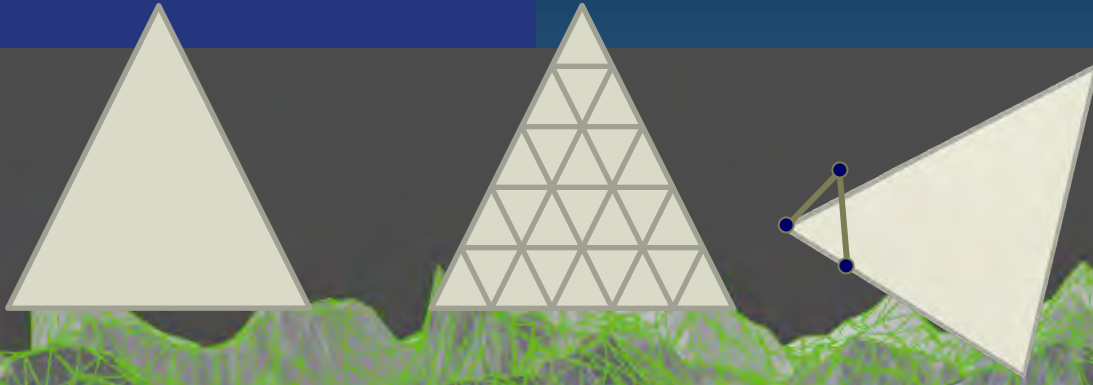
```
static SDL_GLContext maincontext = SDL_GL_CreateContext(window);
```

SDL_GL_CONTEXT_PROFILE_CORE	OpenGL core profile - deprecated functions are disabled
SDL_GL_CONTEXT_PROFILE_COMPATIBILITY	OpenGL compatibility profile - deprecated functions are allowed

OpenGL Evolution

- Controlled by Khronos Group
 - Members include Intel, Nvidia, AMD, Samsung, Sony, ARM, EA, Google...
 - Present version 4.6
 - Evolution reflects new hardware capabilities
 - **More functionality for vertex / fragment programs**
 - **Geometry shaders,**
 - **Tessellation shaders**
 - DX11: Hull shader = GL: Tessellation Control Shader
 - Domain shader = Tessellation Evaluation Shader
 - **Compute shaders (similar to OpenCL, CUDA)**
 - **Mesh shaders**
 - **Ray tracing: GLSL_NV_ray_tracing? or by mixing with OptiX, Vulkan, DirectX 12**
 - Allows for platform specific features through extensions

Tessellation – brief glance



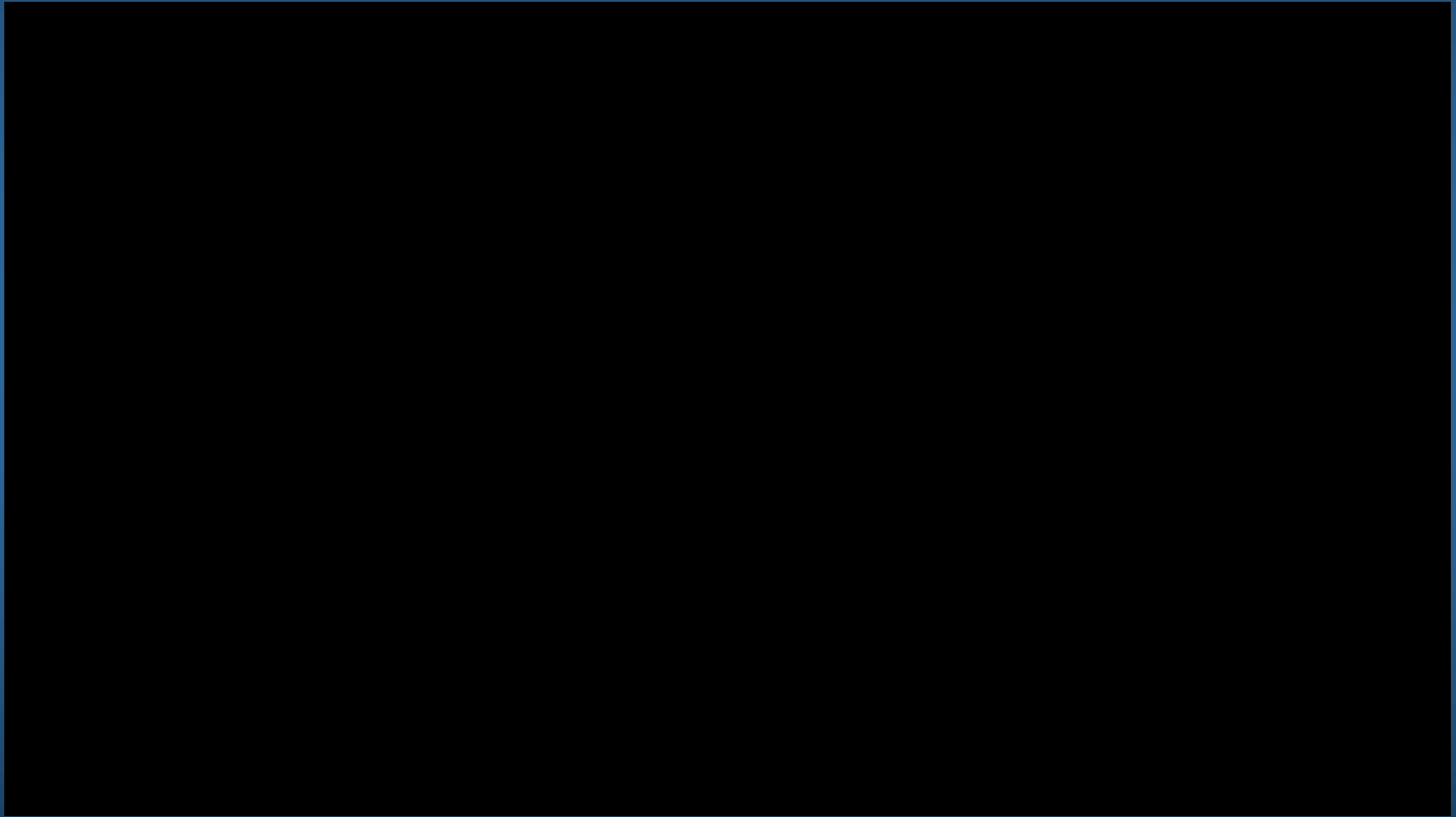
ATI 2007 – Ruby Whiteout



rhinofx



NVIDIA 2010: Endless City



```
#version 400 compatibility
layout(vertices=3) out;
```

```
// thread ID
```

```
#define TID gl_InvocationID
```

```
out float sharable_len[];
```

```
in vec3 eye_space_pos[];
```

```
in vec2
```

```
    scaled_window_space_pos[];
```

```
out vec3 eye_space_pos2[];
```

```
void main(void)
```

```
{
```

```
    sharable_len[TID] =
```

```
        distance(scaled_window_space_pos[TID],
```

```
                scaled_window_space_pos[(TID+1)%3]);
```

```
    barrier();
```

```
    float len0 = sharable_len[0],
```

```
          len1 = sharable_len[1],
```

```
          len2 = sharable_len[2];
```

```
    eye_space_pos2[TID] = eye_space_pos[TID];
```

```
    // Limit level-of-detail output to thread 0
```

```
    if (TID == 0) {
```

```
        // Outer LOD
```

```
        gl_TessLevelOuter[0]=len1; //V1-to-V2 edge
```

```
        gl_TessLevelOuter[1]=len2; //V2-to-V0 edge
```

```
        gl_TessLevelOuter[2]=len0; //V0-to-V1 edge
```

```
        // Inner LOD
```

```
        gl_TessLevelInner[0] =
```

```
            max(len0, max(len1, len2));
```

```
    }
```

```
}
```

Overview of today's OpenGL lecture

- OpenGL
 - Specifying
 - vertices and polygons,
 - vertex arrays to shaders
 - textures
 - Framebuffer Objects
 - Shadow Maps!
 - Blending
 - Misc: point/line width, clip planes
 - Buffers (frame b/f/l/r, depth, stencil)
- Btw, as OS-independent window system (menus, events), you can use:
 - GLUT – The OpenGL Utility Toolkit. But it is old.
 - SDL, GLFW, QT, freeglut (glut does not work for Mac)

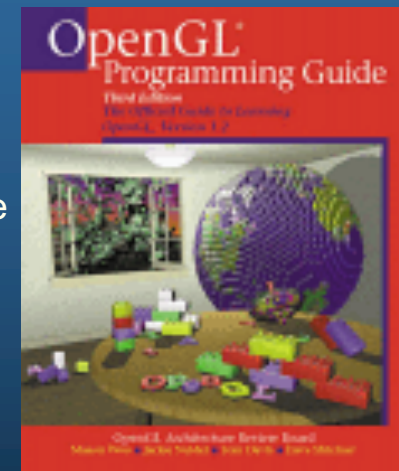
OpenGL – links



- <https://www.khronos.org/files/opengl46-quick-reference-card.pdf>
- Home page: www.opengl.org
- OpenGL 4.6 specification:
 - <https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.core.pdf>
- GLU specification: <http://www.cse.chalmers.se/~uffe/glu1.3.pdf>

You can also find the links on the course home page:
<http://www.cse.chalmers.se/edu/course/TDA362/>

- Programmers Manual and Reference Manual:
 - <http://www.cse.chalmers.se/edu/course/TDA362/redbook.pdf>
 - BUT IT IS HEAVILY OUTDATED BY NOW.
 - Sometimes versions of the RedBook for OpenGL X.X are free online

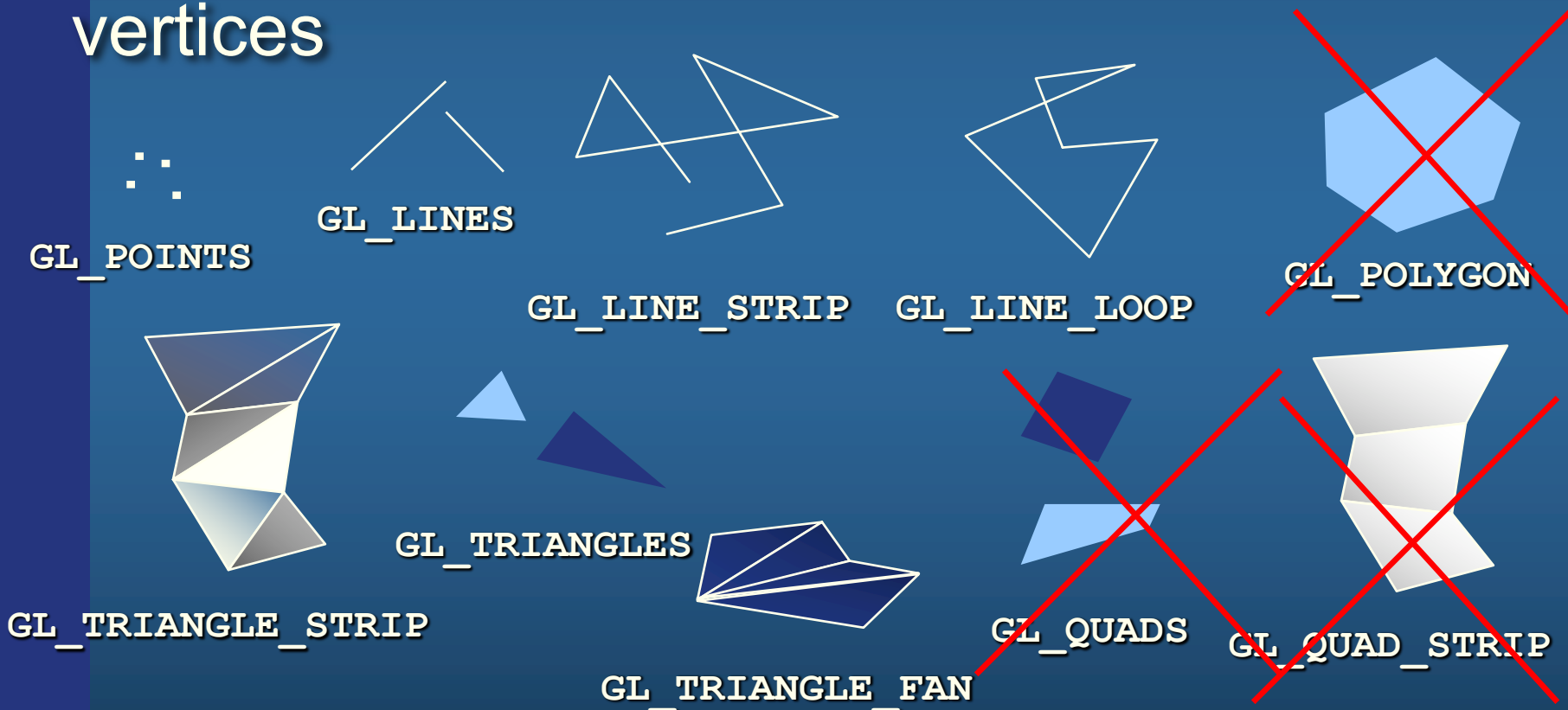


Include

- #include <GL/gl.h>
- Windows:
 - glew.h / glew32.lib / glew32.dll
 - Or (GLEE.h / GLEE.cpp)
 - Link with OpenGL32.lib (MS Windows)
- MacOS:
 - Stopped supporting OpenGL. Might get it to work:
 - Link with -framework OpenGL
 - Now: can install glfw with homebrew and set include- and lib path accordingly and set library name.
 - Video: <https://www.youtube.com/watch?v=MHlbNbWlrIM>
 - But probably does not support OpenGL 4.3 that we use in labs.

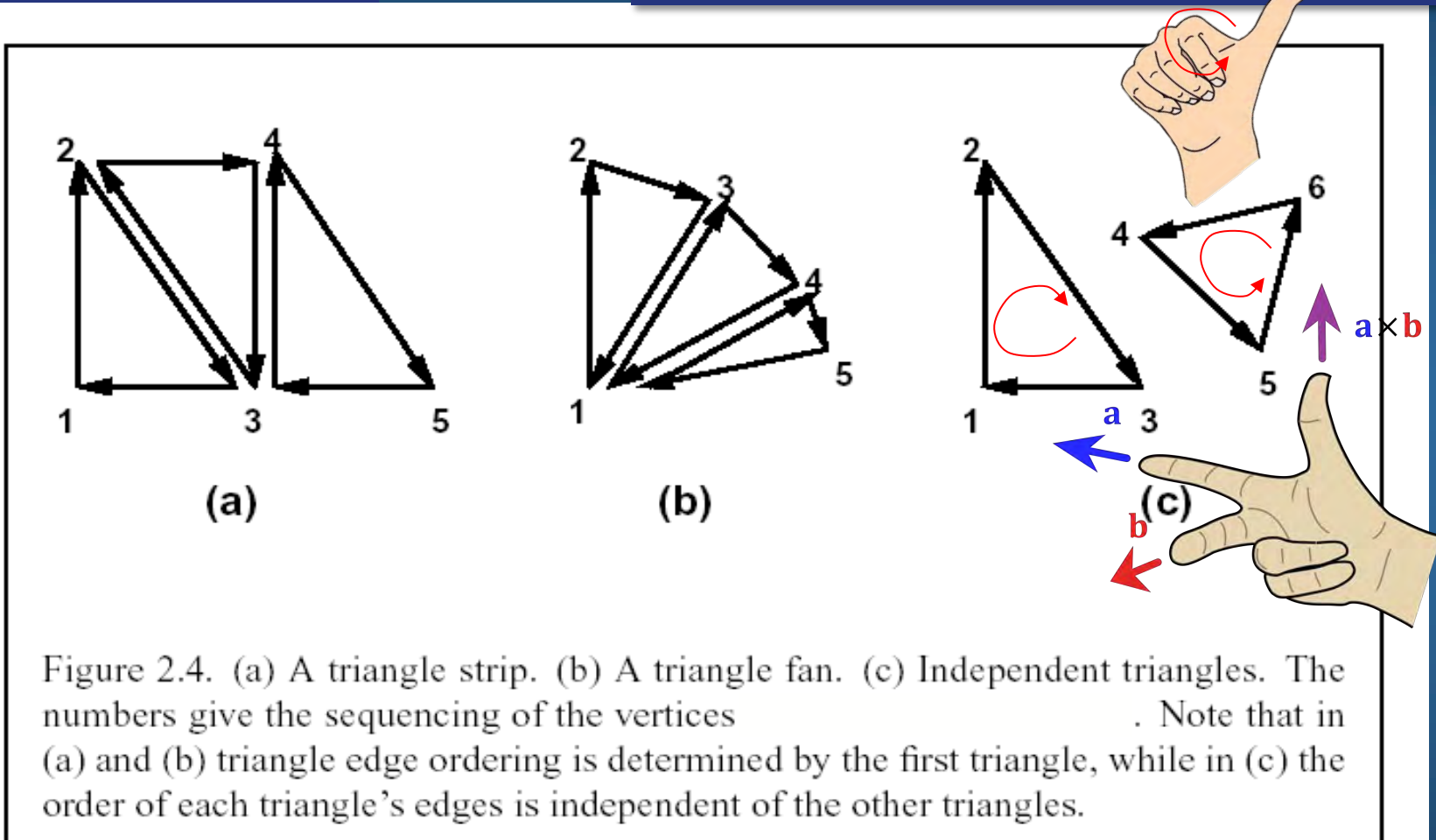
OpenGL Geometric Primitives

- All geometric primitives are specified by vertices



Vertex order

`glFrontFace(enum dir) CCW, CW`
`CullFace(enum mode) -- mode: FRONT, BACK, FRONT_AND_BACK`
`glEnable/Disable(CULL_FACE)`



Note: Vertex order indicates that all but the last triangle is backfacing with CCW-ordering (default for OpenGL).

Specifying vertices and polygons

- OpenGL is a state machine. Commands typically change the current state

Historical Commands:

- Multiple formats for the commands: `void glVertex{234}{sifd}(T coords);`

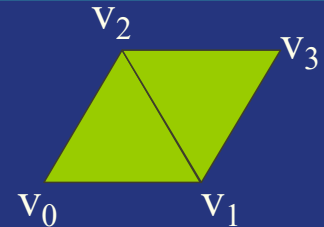
- `glBegin()/glEnd().` (Slow)

```
glBegin(GL_TRIANGLE)
  glVertex3f(0,0,0)
  glVertex3f(0,1,0);
  glVertex3f(1,1,0);
glEnd();
```

Optional: Can specify for instance `glColor3f(r,g,b)`, `glTexCoord2f(s,t)`, `glNormal3f(x,y,z)` - typically per vertex or per primitive.

Index list:

```
vec3f pos[] = v0, v1, v2, v3;
uint indces[] = 0,1,2, 1,3,2;
```



TODAY, WE RATHER USE VERTEX ARRAYS

- Vertex Arrays (Fast):

```
void DrawArrays(enum mode, int first, sizei count);
```

```
void MultiDrawArrays(enum mode, int *first, sizei *count, sizei primcount);
```

```
void DrawElements(enum mode, sizei count, enum type, void *indices);
```

Using index list

other options exist - see the OpenGL Reference Manual online

MultiDrawArrays:

```
for (i = 0; i < primcount; i++)
```

```
  DrawArrays(mode, first[i], count[i]);
```

Example of using Vertex Arrays

1. SEND THE VERTEX COORDINATES TO A BUFFER

```
glGenBuffers( 1, &coordBuffer ); // Create a handle for the coordinate buffer
glBindBuffer( GL_ARRAY_BUFFER, coordBuffer ); // Set the newly created buffer as the current one
glBufferData( GL_ARRAY_BUFFER, sizeof(coords), coords, GL_STATIC_DRAW ); // Send the data
```

// Do the same thing for the color data

```
glGenBuffers( 1, &colorBuffer );
glBindBuffer( GL_ARRAY_BUFFER, colorBuffer );
glBufferData( GL_ARRAY_BUFFER, sizeof(colors), colors, GL_STATIC_DRAW );
```

```
float coords[] = {
// X Y Z
0.0f, 0.5f, 1.0f, // v0
-0.5f, -0.5f, 1.0f, // v1
0.5f, -0.5f, 1.0f, // v2
0.0f, -1.0f, 1.0f // v3
};
```

```
float colors[] = {
// R G B
1.0f, 0.0f, 0.0f, // Red
0.0f, 1.0f, 0.0f, // Green
0.0f, 0.0f, 1.0f, // Blue
1.0f, 1.0f, 0.0f // Yellow
}
```

// Connect triangle data with a Vertex Array Object and the Vertex shader

```
glGenVertexArrays(1, &vertexArrayObject);
glBindVertexArray(vertexArrayObject);
```

// Connects coordBuffer to vertexArrayObject and connects it to Location 0.

```
glBindBuffer( GL_ARRAY_BUFFER_ARB, coordBuffer );
glVertexAttribPointer(0, 3, GL_FLOAT, false/*normalized*/, 0/*stride*/, 0/*offset*/);
```

// Connects colorBuffer to vertexArrayObject and connects it to Location 1.

```
glBindBufferARB( GL_ARRAY_BUFFER_ARB, colorBuffer );
glVertexAttribPointer(1, 3, GL_FLOAT, false/*normalized*/, 0/*stride*/, 0/*offset*/);
```

```
glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);
```

VERTEX SHADER

```
layout(location = 0) in vec3 vertex;
layout(location = 1) in vec3 color;
out vec3 outColor;
uniform mat4 modelViewProjectionMtx;
```

```
void main() {
    gl_Position = modelViewProjectionMtx *
                vec4(vertex, 1);
    outColor = color;
}
```

2. COMMANDS TO DRAW

```
glUseProgram( shaderProgram );
glBindVertexArray(vertexArrayObject);
glDrawArrays( GL_TRIANGLE_STRIP, 0, 4 );
```

Example of a GfxObject Class

```
class GfxObject {  
public:
```

```
    Object() {};  
    ~Object() {};
```

E.g.:

```
    render(mat4 projectionMatrix, mat4 viewMatrix);  
    load("filename");  
    ...
```

```
private:
```

```
    Matrix4x4  
    std::vector<vec3f>  
    std::vector<vec3f>  
    std::vector<vec2f>  
    std::vector<vec3f>
```

```
or just:
```

```
    GLhandle  
    GLuint
```

```
    m_modelMatrix; // Model-to-world matrix  
    m_vertices;  
    m_normals;  
    m_texCoords;  
    m_colors;
```

```
    m_shaderProgram;  
    m_vertexArrayObject;
```

```
{  
    glUseProgram(m_shaderProgram);  
    mat4 modelViewProjectionMatrix = projectionMatrix * viewMatrix *  
                                     m_modelMatrix;  
    int loc = glGetUniformLocation(shaderProgram,  
                                   "modelViewProjectionMatrix");  
    glUniformMatrix4fv(loc, 1, false, &modelViewProjectionMatrix);  
  
    glBindVertexArray(m_vertexArrayObject);  
    glDrawArrays( GL_TRIANGLES, 0, m_vertices.size());  
}
```

The triangle data can be necessary for collision detection and updating of data.

Texture Mapping

You recognize from lab 2

- Steps to add a new texture

- specify texture

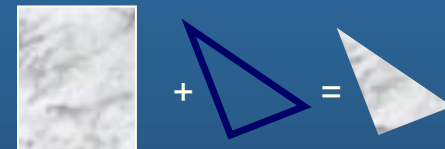
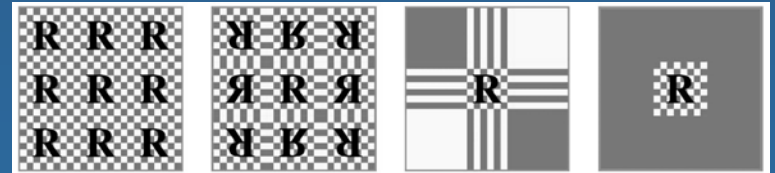
- generate image or read it from file
- assign to texture id:

- `glGenTextures (...);`
`glBindTexture (...);`
`glTexImage2D (...);`
`glGenerateMipMap (...);`

- specify texture parameters

- set texture filter – `glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, ...)`
- set texture wrap mode – `glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, ...)`

- assign texture coordinates to vertices



- When using:

- activate texture unit, bind texture to texture unit, :

- `glActiveTexture(0);`
- `glBindTexture(texID);`

- Do texture lookup in fragment or vertex shader

Texture Mapping

Specifying Texture:

glGenTextures(1, &texID) – generate a texture ID. (Is just an unsigned int.)

glActiveTexture(enum texUnit) -- specify texture unit (up to 32)

glBindTexture(texID), -- specify texture ID that this texture unit and

glTexImage1/2/3D(), **glCopyTexSubImage2D()** -- set / affect image

glGenerateMipmap() -- Create the mipmap hierarchy

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY, ...)

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT)

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)

VERTEX SHADER

```
layout(location = 0) in vec3 vertex;
layout(location = 2) in vec2 texCoordIn;
out vec2 texCoord;
uniform mat4 modelViewProjectionMtx;
void main() {
    gl_Position = modelViewProjectionMtx *
        vec4(vertex, 1);
    texCoord = texCoordIn;
}
```

Specifying Texture Coordinates

1. // Send the TEXTURE COORDINATES to a buffer

```
glGenBuffers( 1, &texcoordBuffer ); // Create a handle for the texcoord buffer
glBindBuffer( GL_ARRAY_BUFFER, texcoordBuffer ); // Set the newly created buffer as the current one
glBufferData( GL_ARRAY_BUFFER, sizeof(texcoords), texcoords, GL_STATIC_DRAW ); // Send the data
```

// Connect texcoord data with the Vertex Array Object and the Vertex shader

```
glBindVertexArray(vertexArrayObject);
```

// Connects texcoordBuffer to vertexArrayObject

```
glBindBuffer( GL_ARRAY_BUFFER_ARB, texcoordBuffer );
glVertexAttribPointer(2, 2, GL_FLOAT, false/*normalized*/, 0/*stride*/, 0/*offset*/);
```

```
glEnableVertexAttribArray(2);
```

glActiveTexture(0); 2. COMMANDS TO DRAW

glBindTexture(texID);

glUseProgram(shaderProgram); glBindVertexArray(vertexArrayObject); glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);

```
float texcoords[] = {
    0.0f, 1.0f,
    0.0f, 0.0f,
    1.0f, 0.0f,
    1.0f, 1.0f
};
```

FRAGMENT SHADER

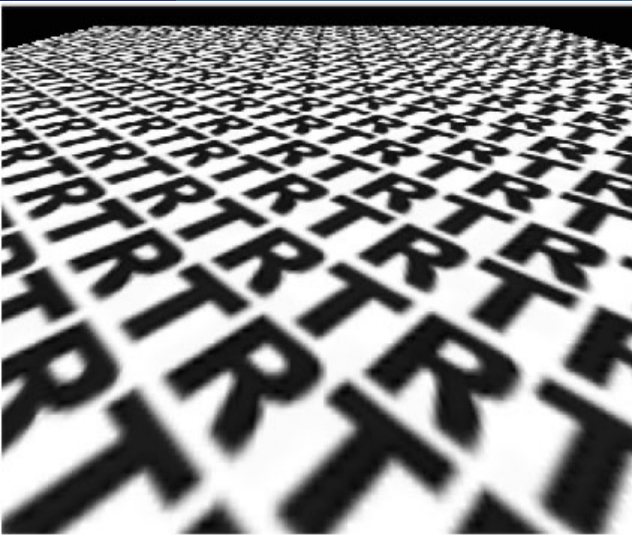
```
layout(binding = 0) uniform sampler2D tex0;
in vec2 texCoord;

void main()
{
    gl_FragColor = texture2D(tex0, texCoord.xy);
}
```

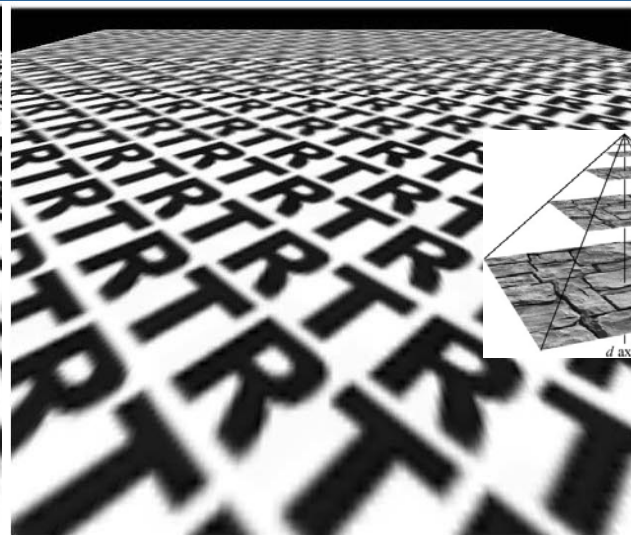
Anisotropic filtering and mipmap generation

Enabling anisotropic filtering:

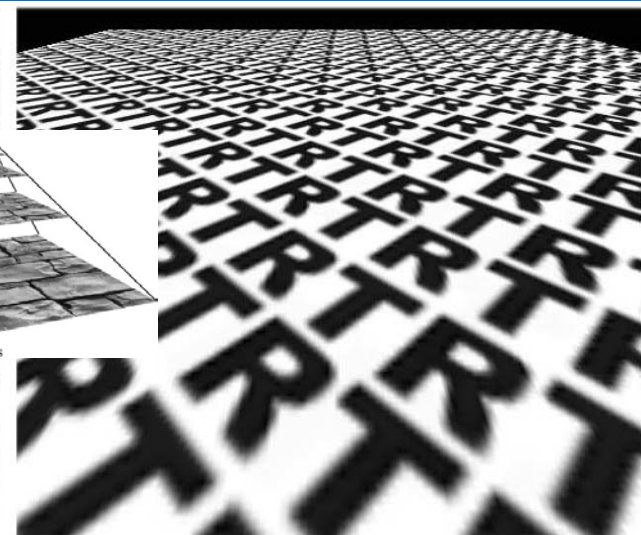
- float MaxAnisotropy
- glGetFloatv(GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT, &MaxAnisotropy);
- glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, MaxAnisotropy);
- glGenerateMipmap(GL_TEXTURE_2D);



21 Nearest/bilinear



Mipmapping



Anisotropic

Specifying a Texture: Other Methods

- Use frame buffer as source of texture image
 - uses current buffer as source image

`glCopyTexImage1D(...)`

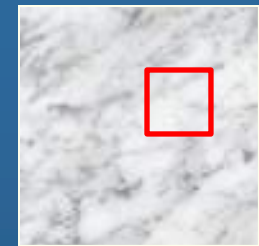
`glCopyTexImage2D(...)`

- Modify part of a defined texture

`glTexSubImage1D(...)`

`glTexSubImage2D(...)`

`glTexSubImage3D(...)`



- Do both with `glCopyTexSubImage2D(...)`, etc.

Reflections with environment mapping

- Texture lookups from an environment map



VERTEX SHADER

```
in vec3    vertex;
in vec3    normalIn;    // The normal
out vec3   normal;
out vec3   eyeVector;

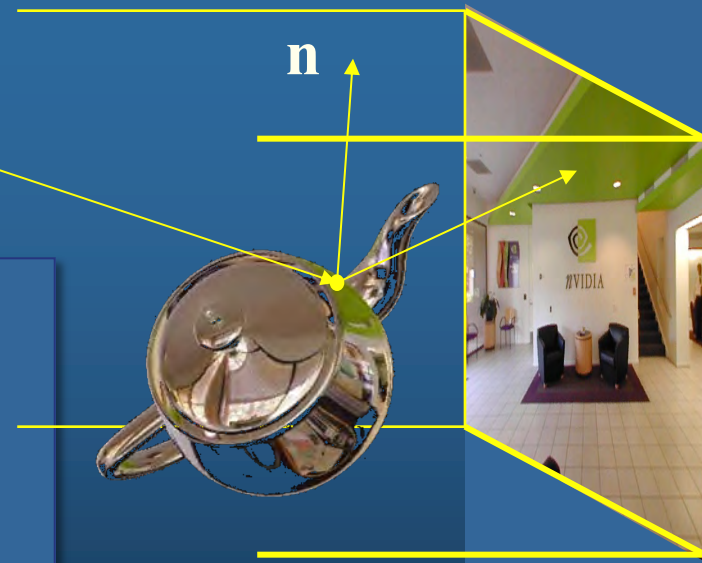
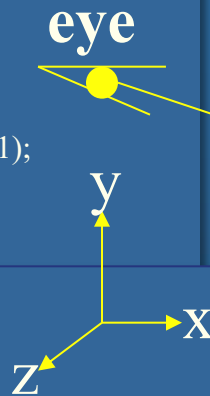
uniform mat4 normalMatrix; //transpose of inverse of modelViewMatrix
uniform mat4 modelViewMatrix;
uniform mat4 modelViewProjectionMatrix;
```

```
void main()
{
    gl_Position = modelViewProjectionMatrix * vec4(vertex,1);
    eyeVector = (modelViewMatrix * vec4(vertex, 1)).xyz;
    normal = (normalMatrix * vec4(normalIn,0.0)).xyz;
}
```

FRAGMENT SHADER

```
in vec3 normal;
in vec3 eyeVector;
uniform samplerCube tex0;
out vec4 fragmentColor;
```

```
void main()
{
    vec3 reflectionVector = normalize(reflect(normalize(eyeVector), normalize(normal)));
    fragmentColor = texture(tex0, reflectionVector);
}
```



Shadow Maps



Shadow Maps

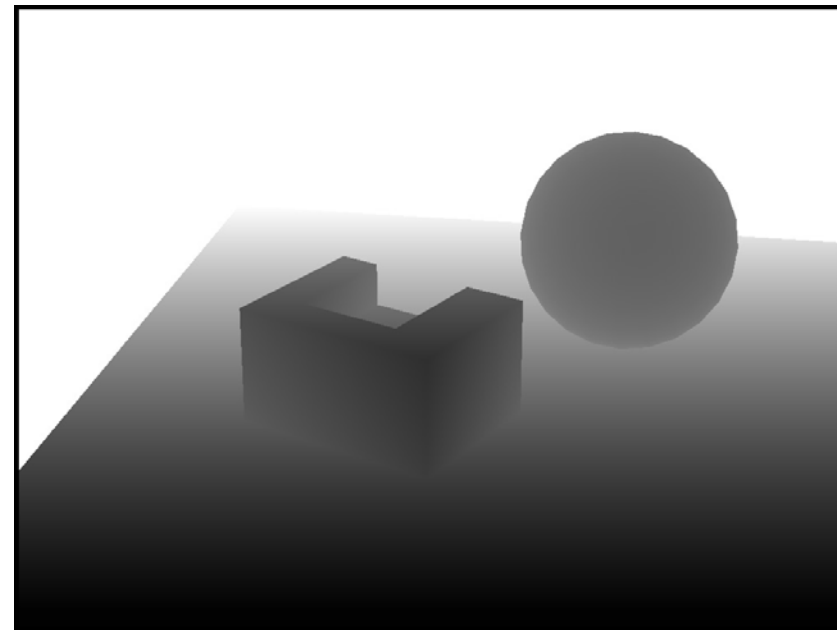
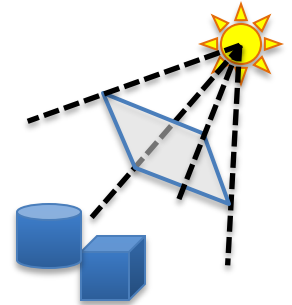


Shadow Maps

Basic Algorithm – the simple explanation:

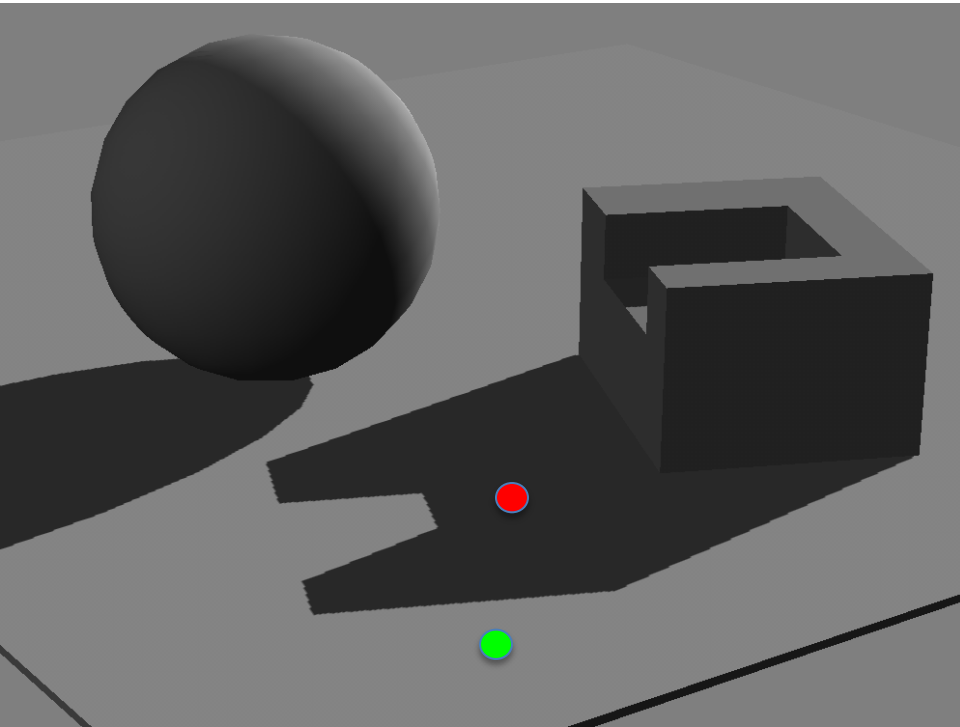
Idea:

- Render image from light source
 - Represents geometry in light
- Render from camera
 - Test if rendered point is visible in the light's view
 - If so -> point in light
 - Else -> point in shadow



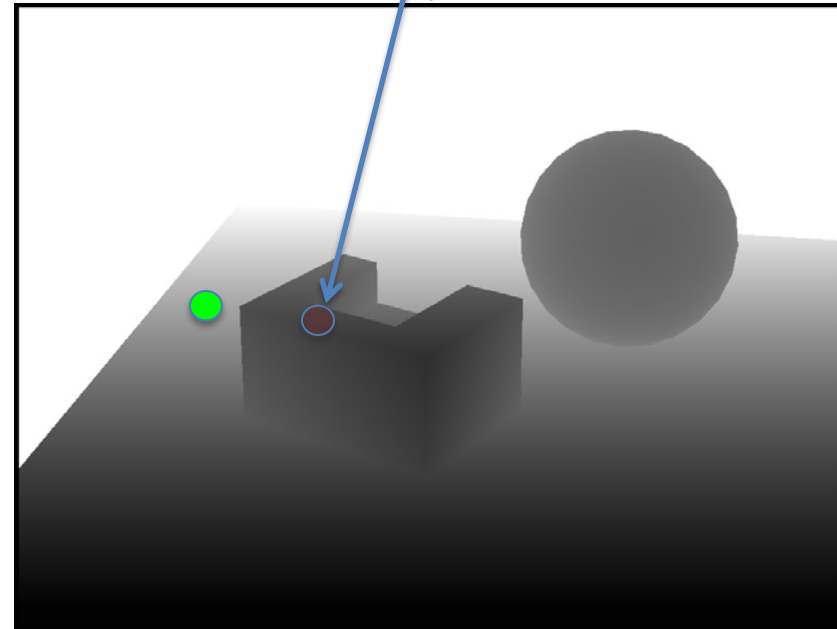
Shadow Map (light's view)

Shadow Maps



Camera's view

Point not represented in shadow map (point is behind box)

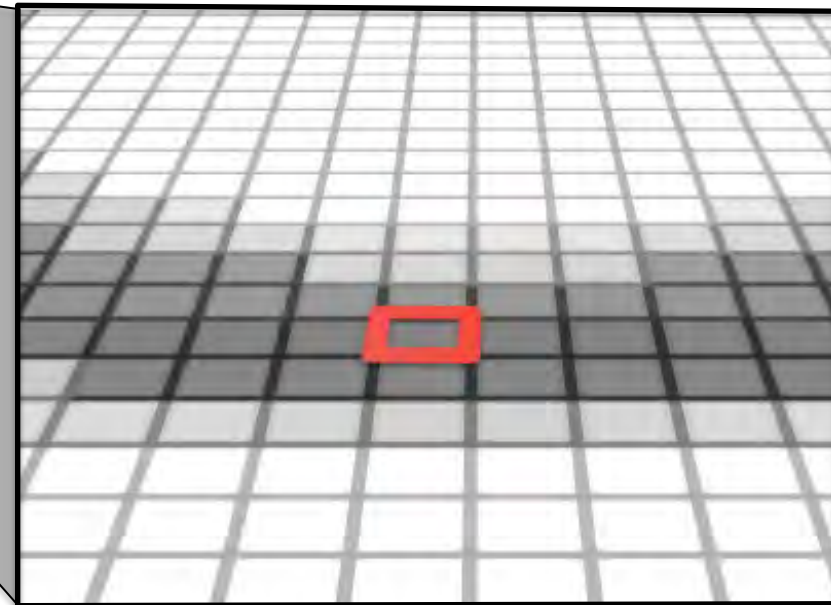
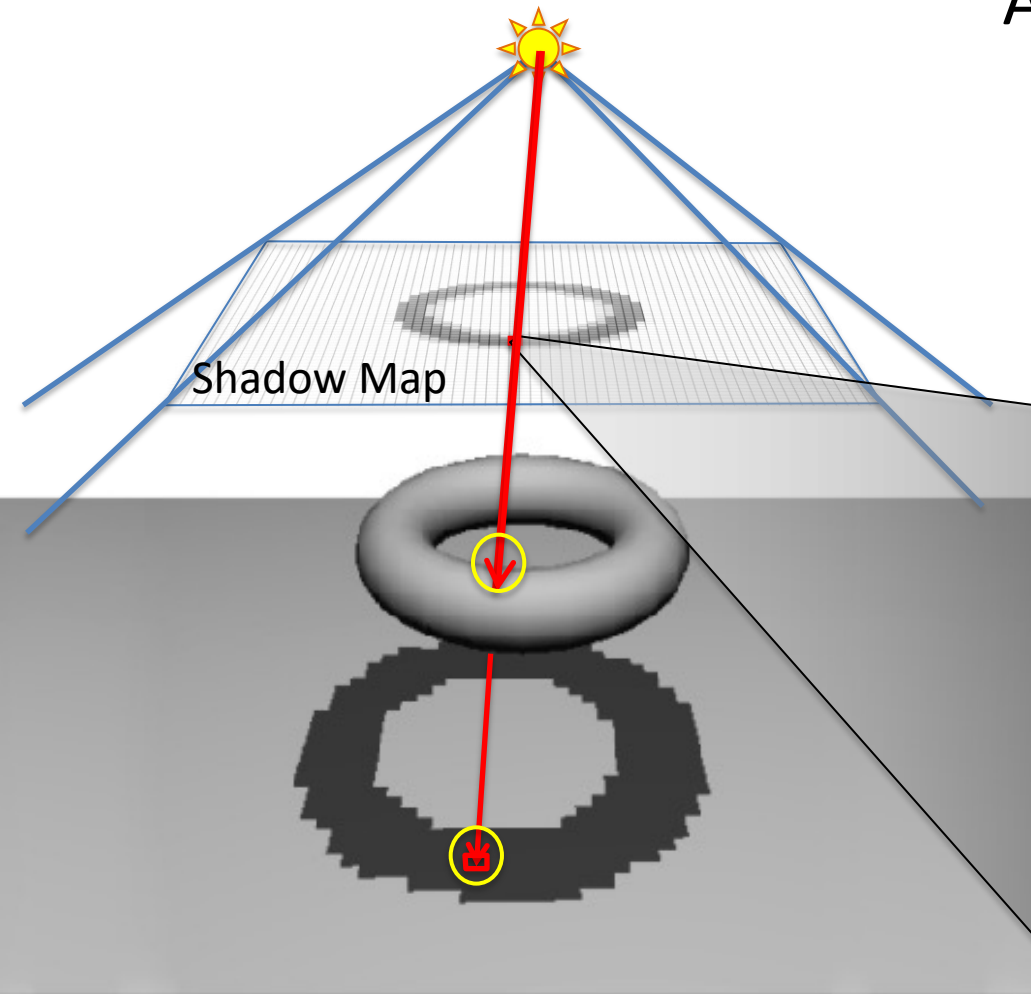


Light's view
(Shadow Map)

Depth Comparison

Render depth image from light

A fragment is in shadow if its depth is greater than the corresponding depth value in the shadow map



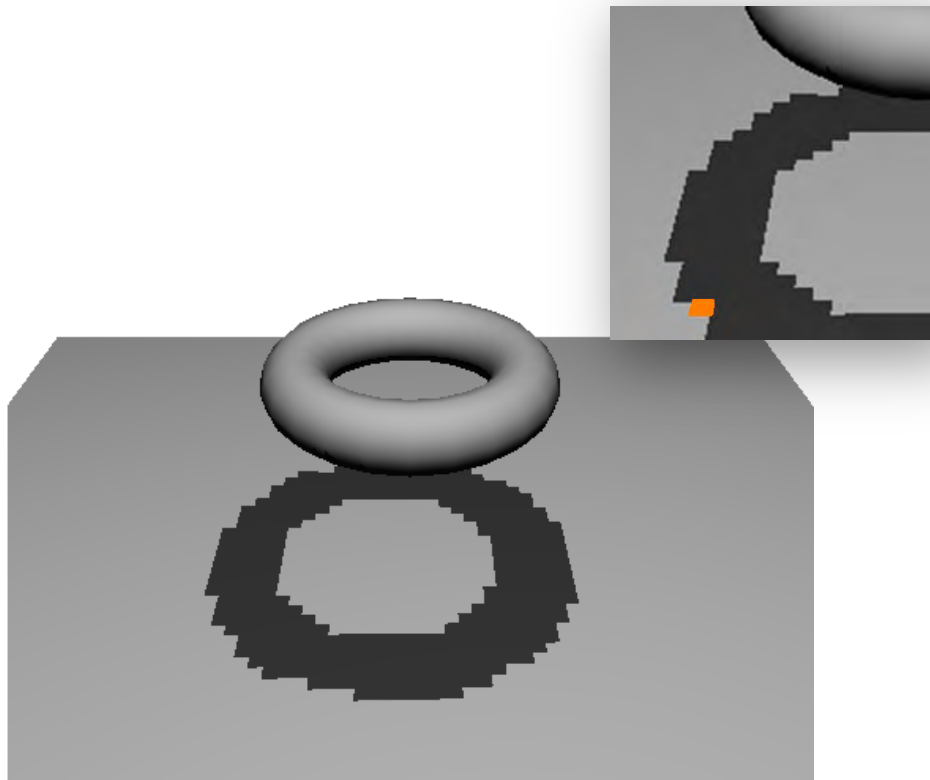
Camera's view

Shadow Maps

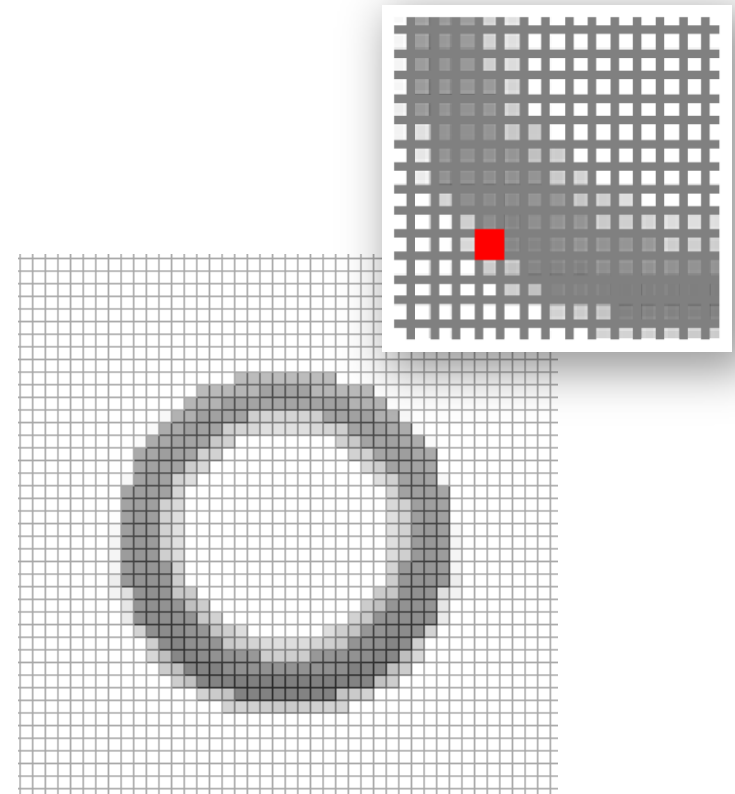
- Pros
 - Very efficient: “This is as fast as it gets”
- Cons...

Shadow Maps - Problems

- Low Shadow Map resolution results in jagged shadows



from viewpoint



from light

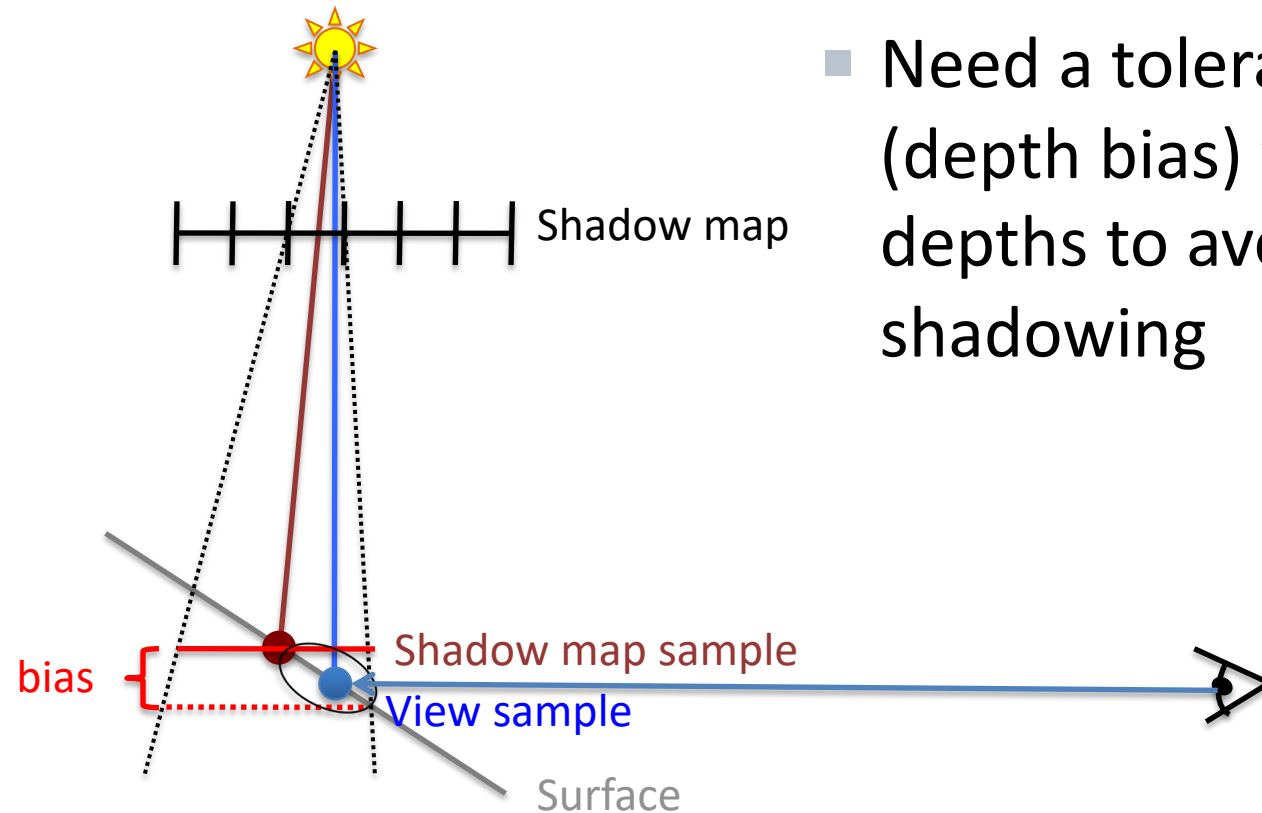
Shadow Maps - Problems

In addition:

- A tolerance threshold (bias) needs to be tuned for each scene for the depth comparison

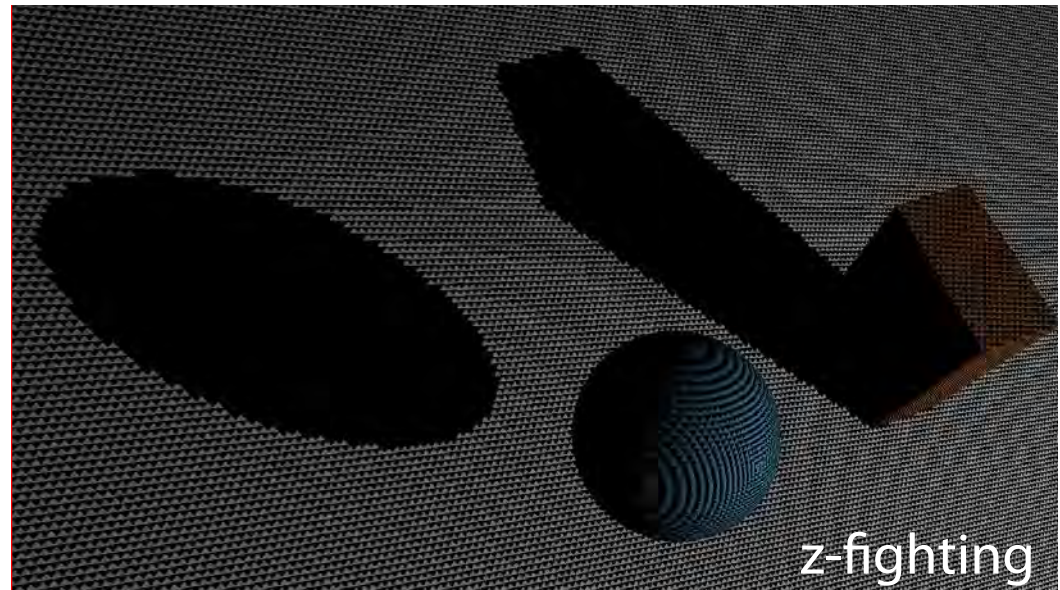
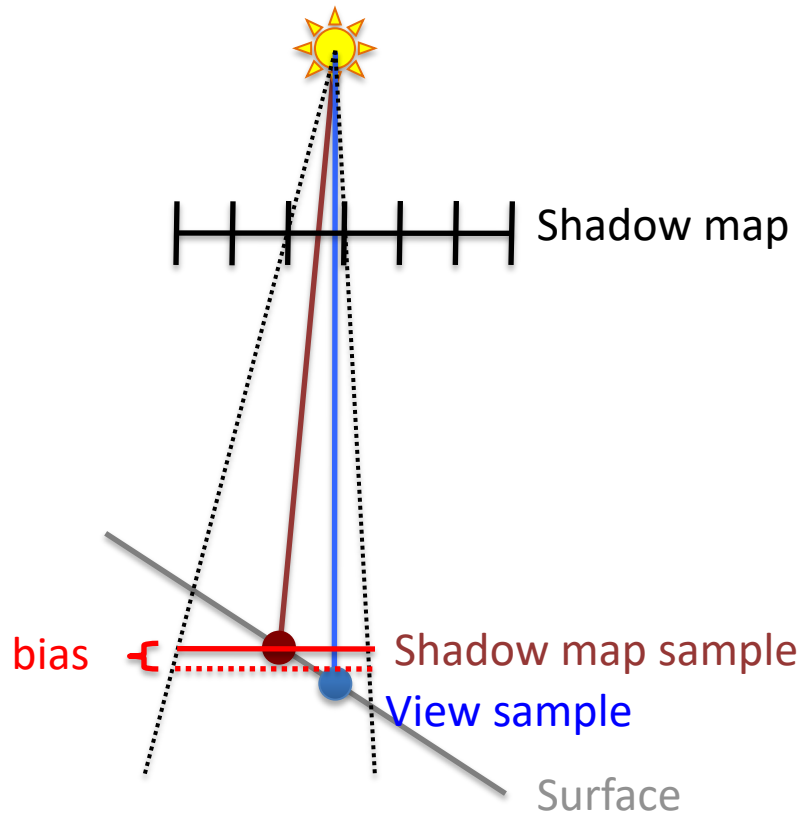
Bias

- Need a tolerance threshold (depth bias) when comparing depths to avoid surface self shadowing



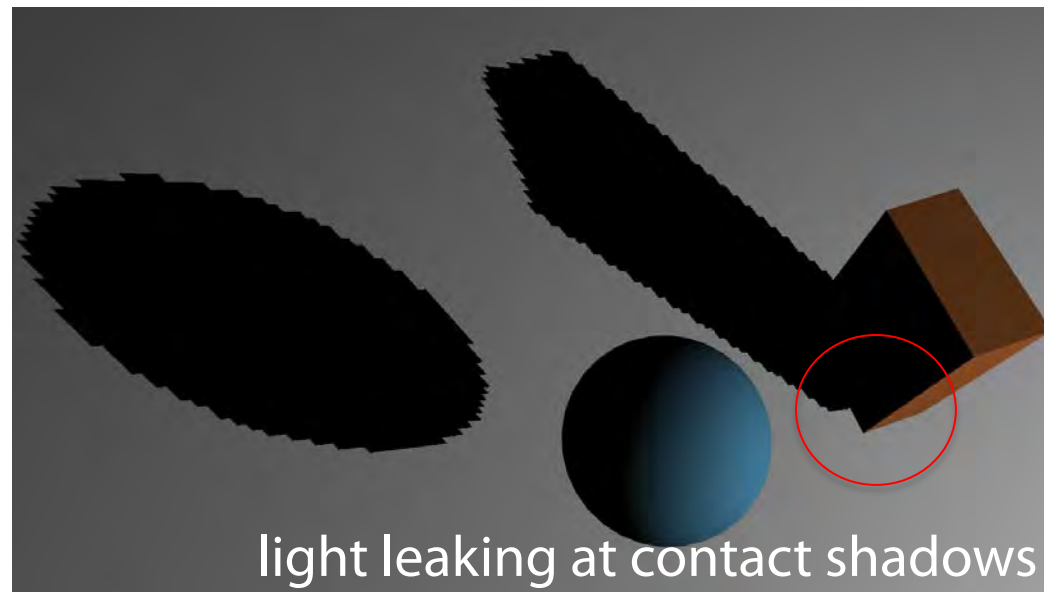
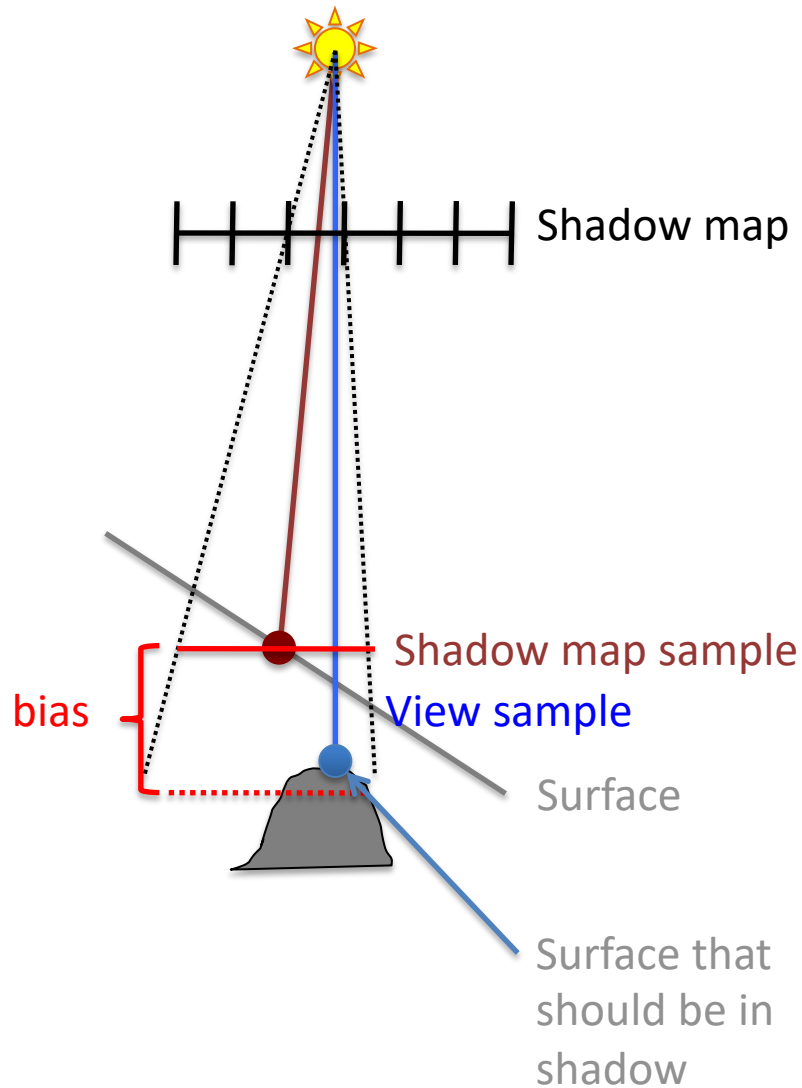
Bias

- Need a tolerance threshold (depth bias) when comparing depths to avoid surface self shadowing



Bias

- Need a tolerance threshold (depth bias) when comparing depths to avoid surface self shadowing



Implementing Shadow Maps

- See tutorial 6 on how to implement shadow maps in practice, as since 2012 😊.
 - Changes every now and then, but algorithm stayed the same since 1978.

Percentage Closer Filtering



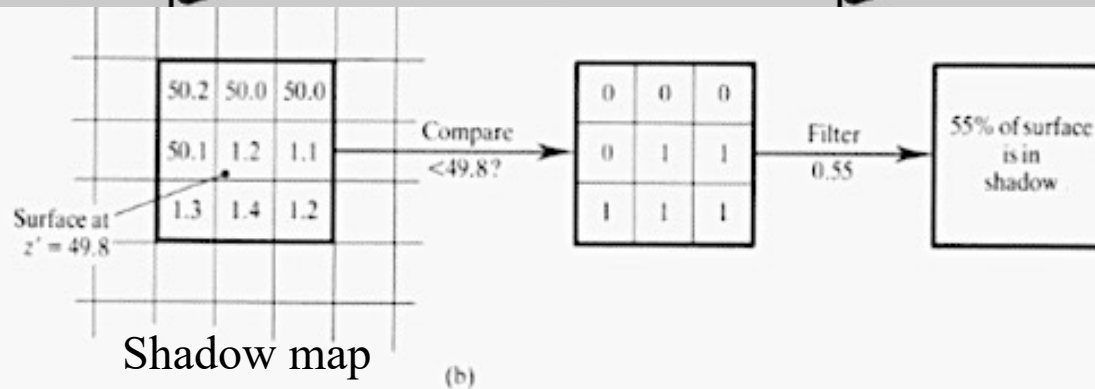
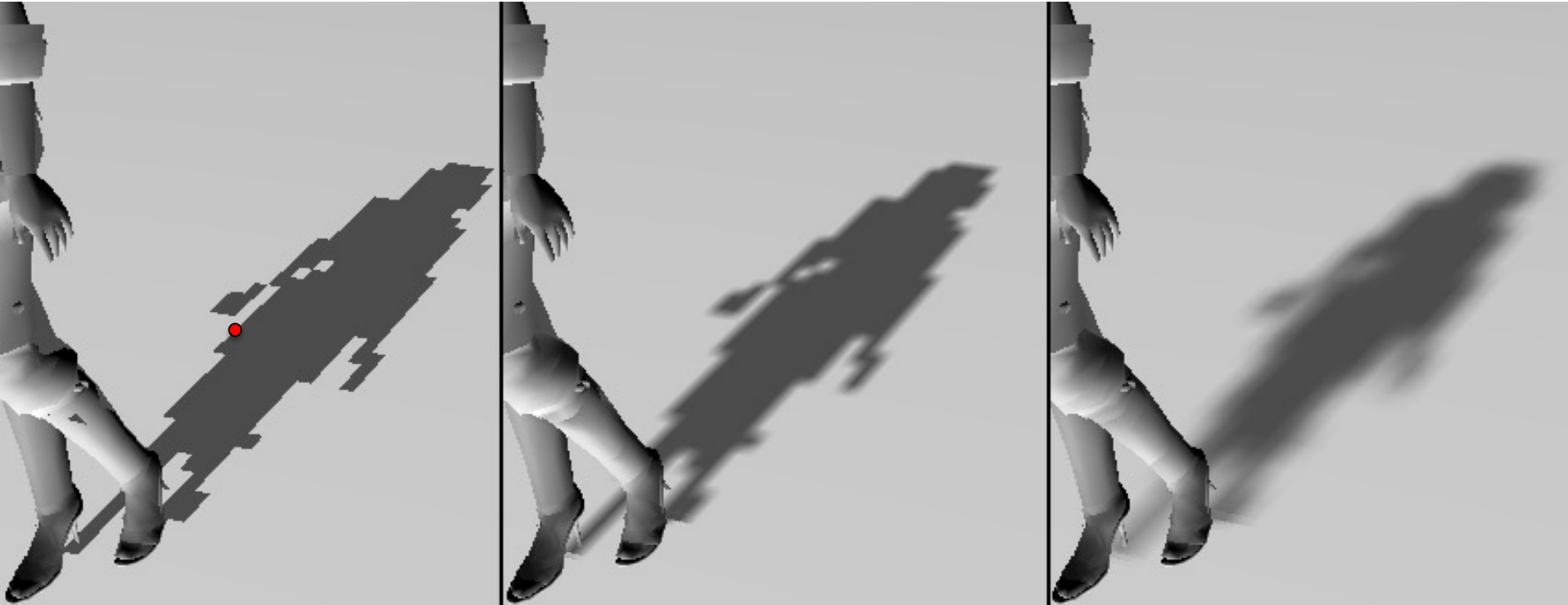
Percentage Closer Filtering

CryTek Soft Shadows



Percentage Closer Filtering

Let's say we make a shadow lookup for this red position. Say the shadow map value is closer to the light (so our position would be in full shadow). Instead, we also check against adjacent shadow-map values, e.g., in a 3x3 or 5x5 neighborhood, and use the average shadow result.



Blending

- Used for

- Transparency

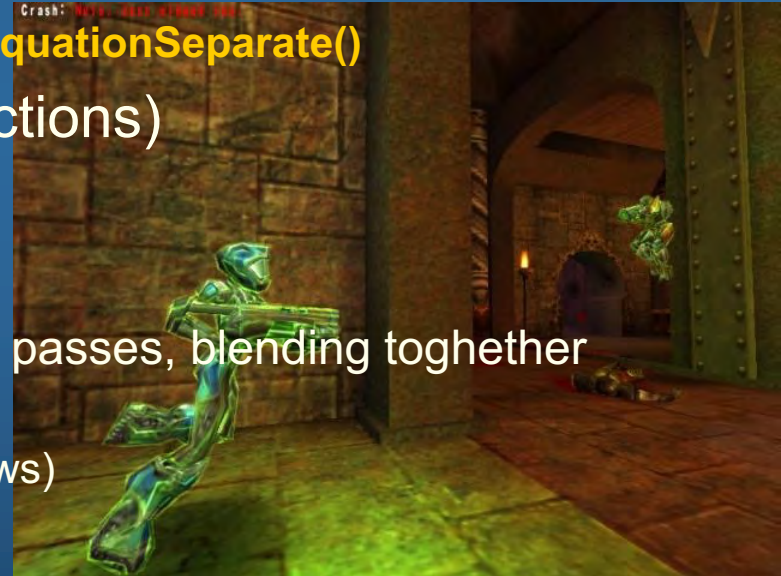
- `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`
 - `glBlendEquation()` // can change + to - or min/max
 - `glBlendFuncSeparate()` / `glBlendEquationSeparate()`

- Effects (particles, planar reflections)

- Before shaders, it was also used for complex materials

- Quake3 uses up to 10 rendering passes, blending together contributions such as:
 - Diffuse lighting (for hard shadows)
 - Bump maps
 - Base texture
 - Specular and emissive lighting
 - Volumetric/atmospheric effects

$$c_o = \alpha c_s + (1 - \alpha) c_d$$



- Enable with `glEnable(GL_BLEND)`

Example of blending for Motion Blur

Possible with usage of e.g blending to 32-bit floating point rgb buffer and averaging result before displaying

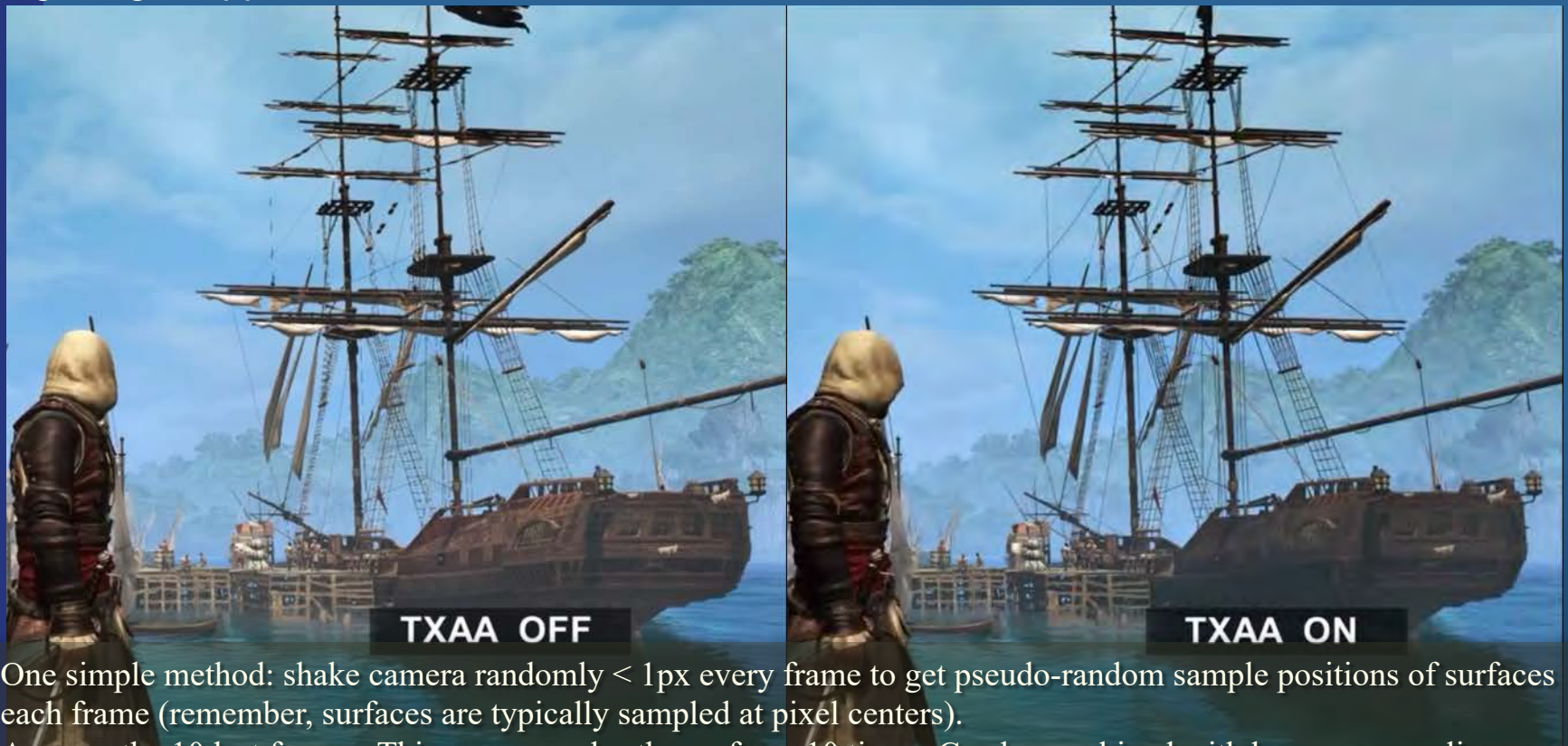


Image courtesy Brostow and Essa

Temporal Anti-aliasing (TXAA)

● Assassin's Creed IV Black Flag

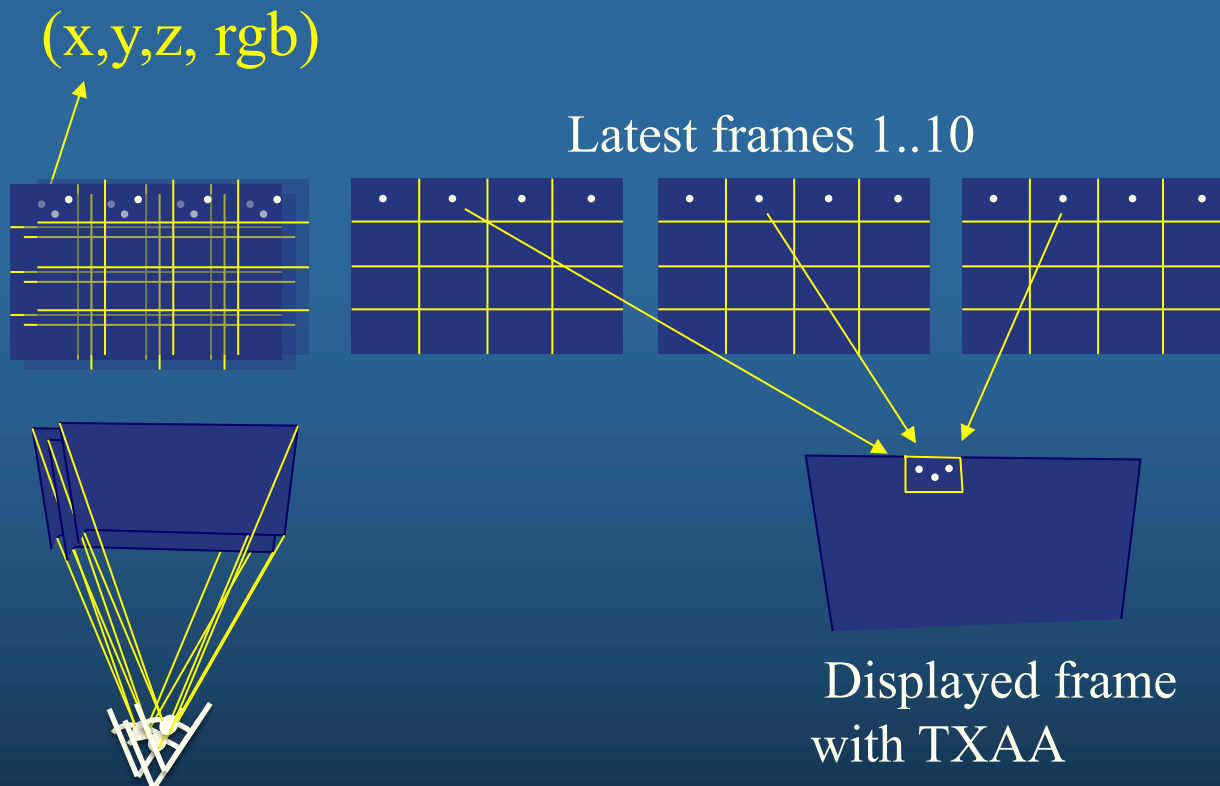
Temporal Anti-aliasing seeks to reduce or remove the effects of **temporal aliasing**. (A common example of **temporal aliasing** in film is the appearance of vehicle wheels travelling backwards, the so-called wagon-wheel effect.) **Temporal anti-aliasing** can also help to reduce jaggies, making images appear softer:



- One simple method: shake camera randomly $< 1\text{px}$ every frame to get pseudo-random sample positions of surfaces each frame (remember, surfaces are typically sampled at pixel centers).
- Average the 10 last frames. This supersamples the surfaces 10 times. Can be combined with hw supersampling.
- For moving objects \rightarrow utilize their motion vectors to fetch correct screen-space sample from frame $(n-i)$, $i \in [0,9]$

Temporal Anti-aliasing (TXAA)

- Reproject the samples from the 10 latest frames to the current frame, by transforming the samples' x, y, z positions from their camera space into the new camera space.



Extra...

- Full screen anti aliasing (FSAA)
 - means super-/multi-coverage- sampling the full screen. Default today.
- FXAA – fast approximate antialiasing, RTR p: 148. [NVIDIA white paper. \(2009\)](#)
- Subpixel Morphological Anti-Aliasing (SMAA)
 - Like FXAA but takes more samples per pixel along edges
- “Filmic SMAA: Sharp Morphological and Temporal Antialiasing” *Siggraph Advances in Real-Time Rendering in Games*, [course notes. \(2016\)](#)
 - Roughly equal to:
 - Edge-detection blur
 - + temporal filtering

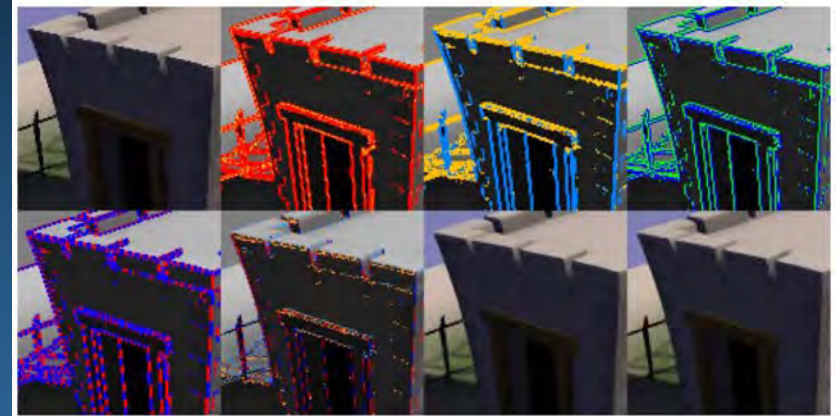


Figure 1: FXAA algorithm from right to left, top to bottom.

Detect the edge directions. Blur each edge orthogonally to its direction.



Misc

Point / Line width

glPointSize(*float size*)

glEnable/Disable(`VERTEX_PROGRAM_POINT_SIZE`)

glLineWidth(*float width*)

glEnable/Disable(`LINE_SMOOTH`)

Polygon rendering

glPolygonMode(*enum face, enum mode*)

– *face*: `FRONT`, `BACK`, `FRONT_AND_BACK`

– *mode*: `POINT`, `LINE`, `FILL`

glPolygonOffset(*float factor, float units*)

glEnable/Disable(*target*)

–`POLYGON_OFFSET_POINT`, `POLYGON_OFFSET_LINE`, `POLYGON_OFFSET_FILL`

Reading Frame Buffers

glReadPixels(*int x, int y, width, height, format, type, void *data*);

glReadBuffer(*enum src*);

–*src*: `NONE`, `FRONT_LEFT`, `FRONT_RIGHT`, `BACK_LEFT`, `BACK_RIGHT`, `FRONT`, `BACK`, `LEFT`, `RIGHT`, `FRONT_AND_BACK`,

–`COLOR_ATTACHMENTi` (where *i* is [0, `MAX_COLOR_ATTACHMENTS` - 1])

glBlitFramebuffer(*srcX0, srcY0, srcX1, srcY1, dstX0, dstY0, dstX1, dstY1, bitfield mask, enum filter*);

–*mask*: Bitwise OR of `COLOR_BUFFER_BIT`, `DEPTH_BUFFER_BIT`, `STENCIL_BUFFER_BIT`

–*filter*: `LINEAR`, `NEAREST`

Buffers

Drawing to Frame Buffers

Selecting a Buffer for Writing :

glDrawBuffer(*enum buf*)

- *buf*: NONE, FRONT_LEFT, FRONT_RIGHT, BACK_LEFT, BACK_RIGHT, FRONT, BACK, LEFT, RIGHT, FRONT_AND_BACK, COLOR_ATTACHMENT*i* (where *i* is [0, MAX_COLOR_ATTACHMENTS - 1]),
- AUX*i* (where *i* is [0, AUX_BUFFERS - 1])

DrawBuffers(*sizei n*, const *enum *bufs*);

- *bufs*: NONE, FRONT_LEFT, FRONT_RIGHT, BACK_LEFT, BACK_RIGHT,
- COLOR_ATTACHMENT*i* (where *i* is [0, MAX_COLOR_ATTACHMENTS - 1]),
- AUX*i* (where *i* is [0, AUX_BUFFERS - 1])

FRAGMENT SHADER

```
layout(location = 0) out vec4 fragColor0;
layout(location = 1) out vec4 fragColor1;
void main()
{
    fragColor0 =vec4(1,0,0,1);
    fragColor1 =vec4(1,1,0,1);
}
```

Framebuffer Objects

Binding & Managing Framebuffer Objects (collection of renderbuffers, (<=8 colbuffs))

- **glBindFramebuffer(), glGenFramebuffers(), glDeleteFramebuffers()**

Renderbuffers:

- **BindRenderbuffer(), DeleteRenderBuffers(), glGenRenderBuffers(), glRenderBufferStorage() – w,h,depth/color/stencil**

Attaching renderbuffer to current framebuffer object

- **glFramebufferRenderbuffer()**

Attaching Texture Image to Framebuffer (i.e., render-to-texture)

- **glFramebufferTexture1/2/3D()**

Buffers

- Frame buffer
 - `glColorMask(GLboolean red, green, blue, alpha);`
 - `glColorMaski(Gluint buf, GLboolean red, green, blue, alpha);`
- Depth buffer (z-buffer)
 - For correct depth sorting
 - (instead of BSP-algorithm, painters algorithm...)
 - `glDepthFunc()`, – `GL_LESS`, `GL_LEQUAL`, `GL_GREATER`...
 - `glDepthMask(false)` – disables writing of z-values into depth buffer
 - `glDisable(GL_DEPTH)` – disables depth testing
- Stencil buffer (e.g. used for Shadow volumes)
 - `glStencilFunc(GL_EQUAL, 0, 0xffffffff)`
 - enables and disables color-/depth-buffer drawing on a per-pixel basis – here only where stencilbuffer = 0.
 - less, lequal, greater, gequal, always – based on stencil-buffer value
 - `glStencilFuncSeparate(...)` – settings for back-facing vs front-facing polygons
 - `glStencilOp` – how to update the stencil buff on stencil+z-test: keep, replace, incr, decr.
 - `(glStencilMask` – enable/disable writing of individual bits in the stencil buffer)
- General commands:
 - `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT)`
 - Specify clearing value:, `glClearStencil()`, `glClearColor()`, `glClearDepth(default=1)`

Specials



- "Clip planes" (8):
 - Fragment shader: `glClipDistance[]`
 - which are sent as outputs from the vertex shader to the fragment shader.
 - `glEnable(GL_CLIP_DISTANCEi)`
- Scissors:
 - `glScissor(x,y,w,h)`, `glEnable(GL_SCISSOR_TEST)`
- Finishes all draw calls before CPU-execution continues:
 - `glFinish()`

Fragment Operations

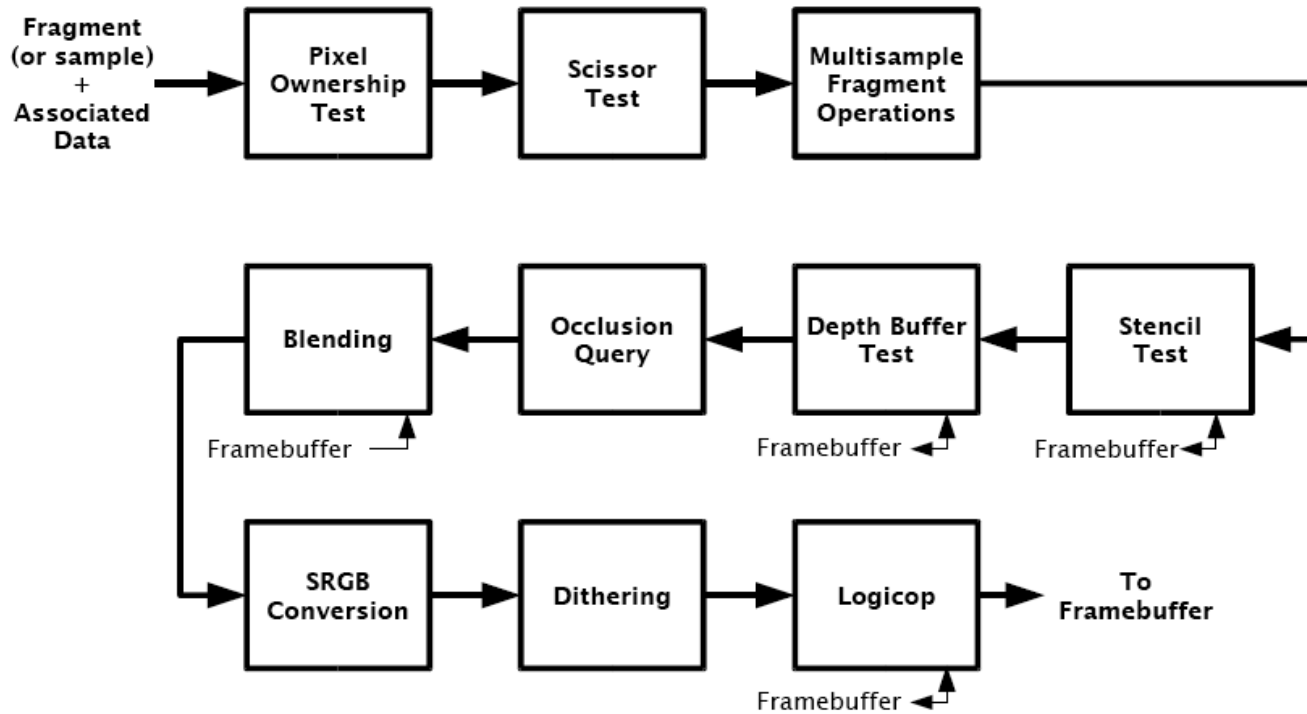


Figure 4.1. Per-fragment operations.

Other misc. newer functionality

- *Image* (“object/buffer/texture”) OpenGL 4.2

in Fragment shader to read/write to/from an *image* texture:

- `gvec4 imageLoad(gimage image, vec2 pos);`
- `void imageStore(gimage image, vec2 pos, gvec4 data);`
- https://www.khronos.org/opengl/wiki/Image_Load_Store

- *Transform feedback*

- Routing vertex-processing (vertex-, tessellation-, geometry shaders) results into a buffer object on the GPU instead of to the rasterization stage.
- `void glBeginTransformFeedback(GLenum primitiveMode);`
- `void glEndTransformFeedback(void);`

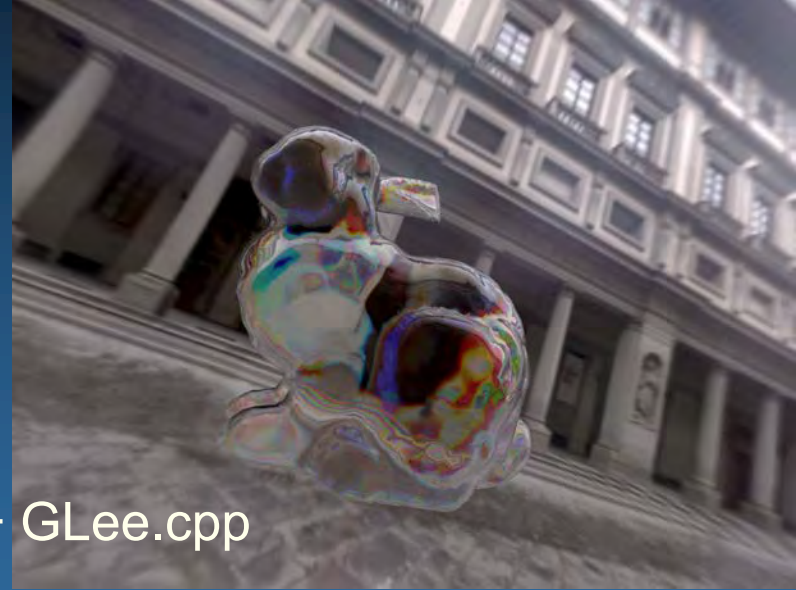
- *Occlusion queries*

to count #pixels drawn to in the frame buffer.

- `glBeginQuery(GLenum target, GLuint id);`
- `glEndQuery(GLenum target);`

Extensions

- `glew.h + glew32.lib/dll` OR `GLee.h + GLee.cpp`
- Or get the extensions manually:
- Check if extension is supported:
`glutExtensionSupported("GL_EXT_framebuffer_sRGB")`
`glutExtensionSupported("GL_EXT_texture_integer")`
- Get address of extension function:
 - `glTexParameterIivEXT = wglGetProcAddress("glTexParameterIivEXT");`
 - `glClearColorIiEXT = wglGetProcAddress("glClearColorIiEXT");`



Types of Exam Questions

- principles of a real-time rendering API like OpenGL
 - E.g. high level functionality
 - Shadow Maps
 - Types of buffers
 - How do you achieve transparency?
 - What defines what is the back and front side of a triangle?

Hunter ate Ranger's rocket
Wrote screenshots/shot0147.tga

END OF OPENGL LECTURE

