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.

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
  - (Now after many upgrades very similar to OpenGL)
- OpenGL
  - SGI
  - Historically:
    - "Precede the hardware evolution"
    - Operation system independent
    - Window system independent
    - Industry, games (Quake –John Carmack, Apple, Linux)
    - January 1992
    - Extendable, stable, better design,

Direct3D was messy to program version 3.0 – 6.0.

Today version 12

Today version 4.7

OpenGL – simplicity

- Single uniform interface to different 3D accelerators
- Hide different capabilities, requiring full support of the whole OpenGL feature set (using software emulation if necessary)

```
glMatrixMode( GL_PROJECTION );     /* Subsequent matrix commands will affect the projection matrix */
glLoadIdentity();                  /* Initialise the projection matrix to identity */
glFrustum( -1, 1, -1, 1, 1, 1000 ); /* Apply a perspective-projection matrix */

glMatrixMode( GL_MODELVIEW );      /* Subsequent matrix commands will affect the modelview matrix */
glLoadIdentity();                  /* Initialise the modelview to identity */
glTranslatef( 0, 0, -3 );          /* Translate the modelview 3 units along the Z axis */


glBegin( GL_POLYGON );   /* Begin issuing a polygon */
glColor3f( 0, 1, 0 );    /* Set the current color to green */
glVertex3f( -1, -1, 0 ); /* Issue a vertex */
glVertex3f( -1, 1, 0 );  /* Issue a vertex */
glVertex3f( 1, 1, 0 );   /* Issue a vertex */
glVertex3f( 1, -1, 0 );  /* Issue a vertex */
glEnd();                 /* Finish issuing the polygon */
```

E.g., getting OpenGL 4.1

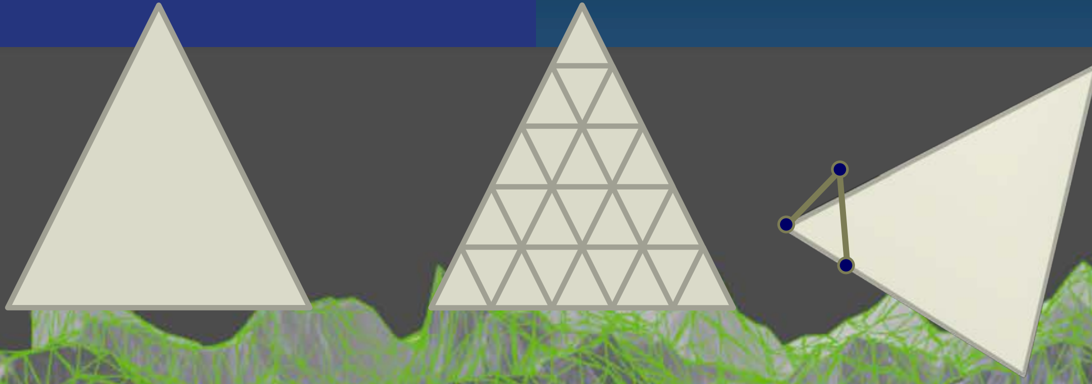
```
SDL_GL_LoadLibrary(nullptr); // Default OpenGL is fine.

// 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 ony 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);
```

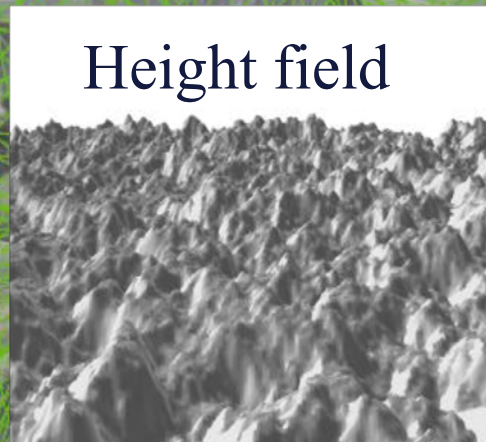
SDL_GL_CONTEXT_PROFILE_CORE	OpenGL core profile - deprecated functions are disabled
SDL_GL_CONTEXT_PROFILE_COMPATIBILITY	OpenGL compatibility profile - deprecated functions are allowed

OpenGL Evolution

- Controlled by an Architecture Review Board (ARB)
  - Members include Apple, Intel, Nvidia, AMD, Samsung, Sony, ARM, Epic Games, Google…
  - Present version 4.5
    - Evolution reflects new hardware capabilities
      - **More functionality for vertex / fragment programs**
      - **Geometry shaders,**
      - **Tesselation units**
        - DX11: Hull shader = GL: Tesselation Control Shader
        - Domain shader = Tesselation Evaluation Shader
  - Allows for platform specific features through extensions

Tesselation – brief glance





- Input Assembler
- Vertex Shader
- Hull Shader
- Tessellator
- Domain Shader
- Geometry Shader
- Rasterizer
- Pixel Shader
- Output Merger

GLSL Tessellation Control Shader


```
#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

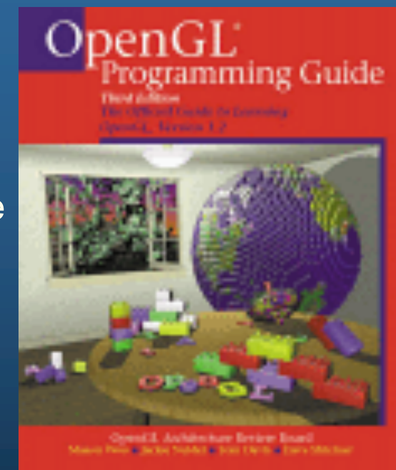
- OpenGL
  - Specifying vertices and polygons, Buffer Objects
  - Shaders
  - Framebuffer Objects
  - Texturing
  - Shadow Maps!
  - Blending
  - Buffers (frame b/f/l/r, depth, alpha-channel, stencil)
  - Misc: point/line width, clip planes
- GLU – The OpenGL Graphics System Utility Library
- GLUT – The OpenGL Utility Toolkit
  - Windows and menus
  - Callbacks for events
  - Text support
  - Predefined Objects

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



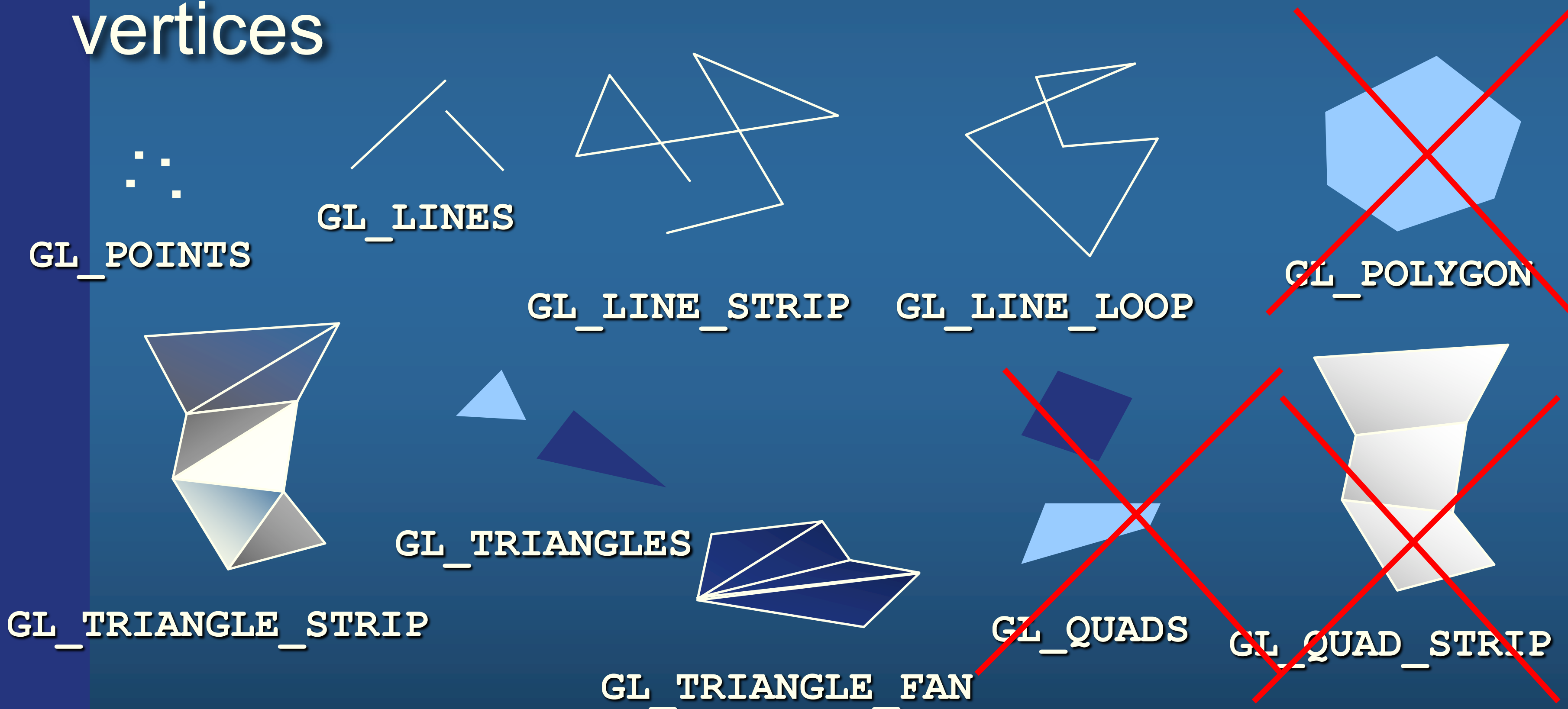
Include

- #include <GL/gl.h>
- Links with OpenGL32.lib (MS Windows)

- glew.h / glew32.lib / glew32.dll
- (GLee.h / GLee.cpp)

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)

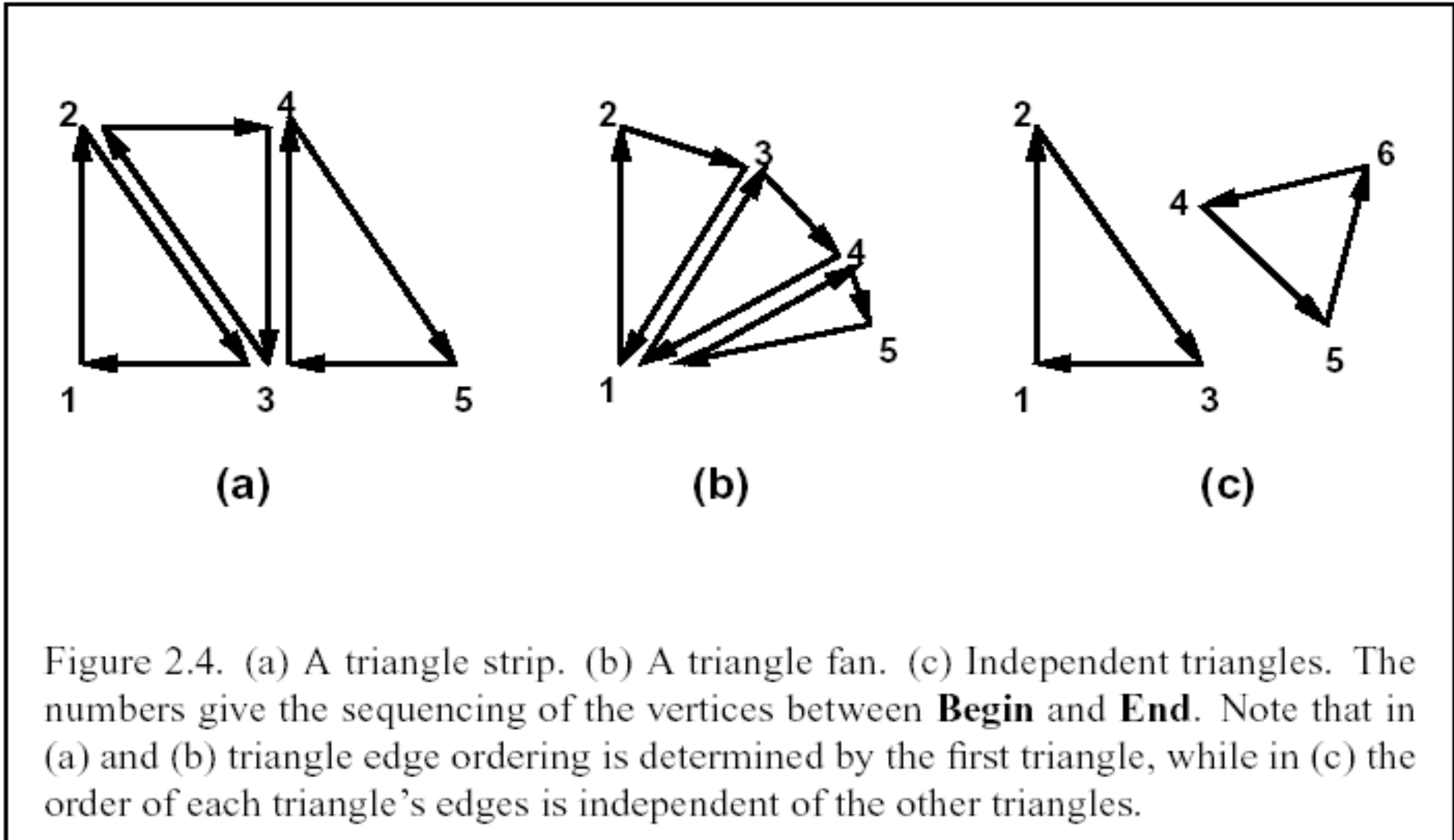


Figure 2.4. (a) A triangle strip. (b) A triangle fan. (c) Independent triangles. The numbers give the sequencing of the vertices between **Begin** and **End**. Note that in (a) and (b) triangle edge ordering is determined by the first triangle, while in (c) the order of each triangle's edges is independent of the other triangles.

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.

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 );
```

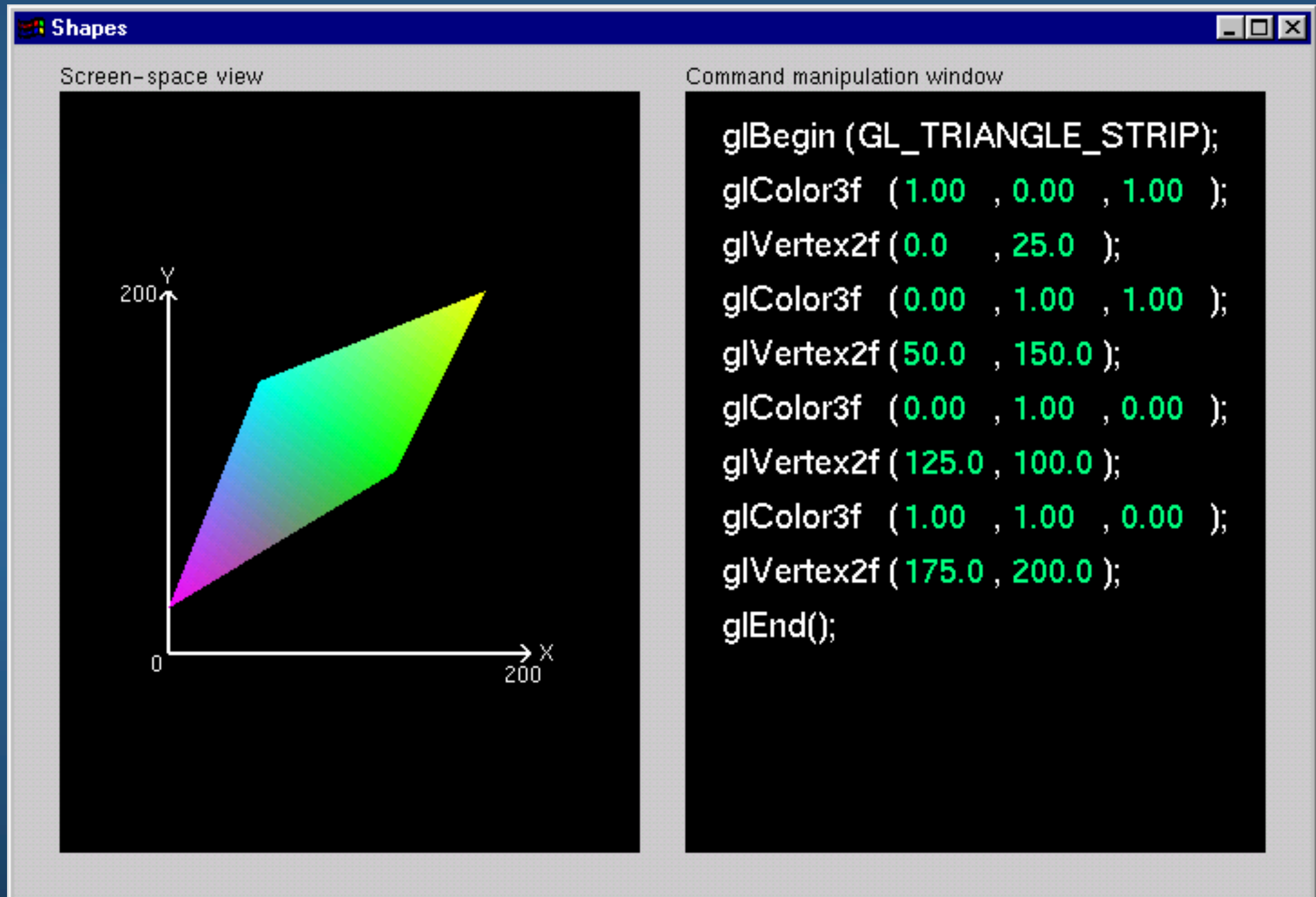
```
Float coords[] = {
//  X     Y    Z
  0.0f,  0.5f, 1.0f,   // v0
 -0.5f, -0.5f, 1.0f,   // v1
  0.5f, -0.5f, 1.0f,   // v2
  0.0f, -1.0f, 1.0f    // v3
};

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 activates coordBuffer for next command below.
glBindBuffer( GL_ARRAY_BUFFER_ARB, coordBuffer );
glVertexAttribPointer(0, 3, GL_FLOAT, false/*normalized*/, 0/*stride*/, 0/*offset*/ );

// Connects colorBuffer to vertexArrayObject and activates colorBuffer for command below.
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 historical slow way:



Example of a GfxObject Class

```
class GfxObject {
public:
        Object() {};
        ~Object() {};

        render(mat4 projectionMatrix, mat4 viewMatrix);
        load("filename");
        …
private:
        Matrix4x4              m_modelMatrix;  // Model-to-world matrix
        std::vector<vec3f>     m_vertices;
        std::vector<vec3f>     m_normals;
        std::vector<vec2f>     m_texCoords;
        std::vector<vec3f>     m_colors;
or just:
        GLhandle               m_shaderProgram;
        GLuint                 m_vertexArrayObject;
}
```

E.g.:

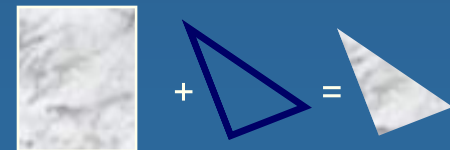
```
{
  glUseProgram(m_shaderProgram);
  mat4 modelViewProjectionMatrix = projectionMatrix * viewMatrix *
                                   m_modelMatrix;
  int loc = glGetUniformLocation(shaderProgram,
                    "modelViewProjectionMatrix");
  glUniformMatrix4fv(loc, 1, false, &modelViewProjectionMatrix);

  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

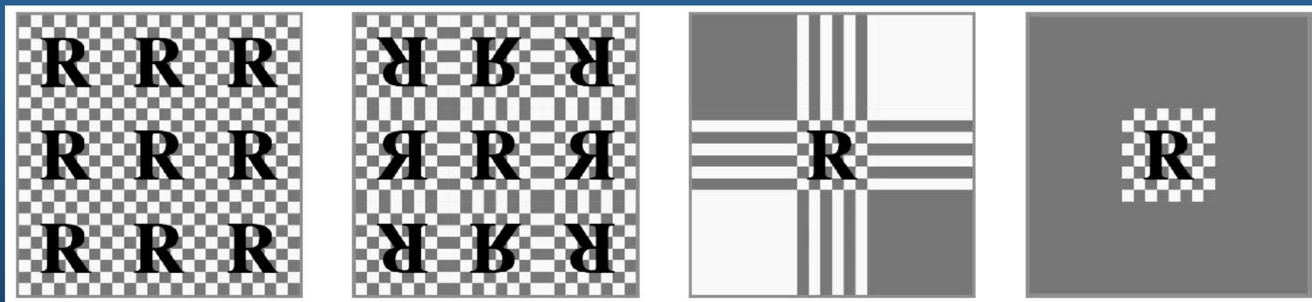
You recognize from lab 2

- Three steps
  ① specify texture
    - generate image or read it from file
    - assign to texture – `glGenTextures(), glBindTexture(), glTexImage2D(), glGenerateMipMap()`

  ② assign texture coordinates to vertices

  ③ 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,…)`



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 an
- **glTexImage1/2/3D (), glCopyTexSubImage2D()** -- set / affect imag
- **glGenerateMipMap()** -- Create the mipmap hierarchy

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPE
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPE
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LI
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

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

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);
```

FRAGMENT SHADER

```
layout(binding = 0) uniform sampler2D tex0;

in vec2 texCoord;

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

2. COMMANDS TO DRAW

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

Example of Loading a Texture

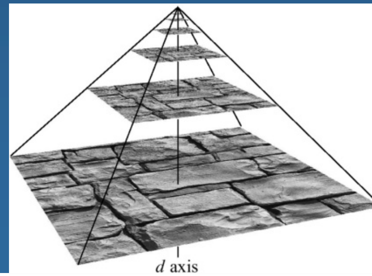
Do once when loading texture:

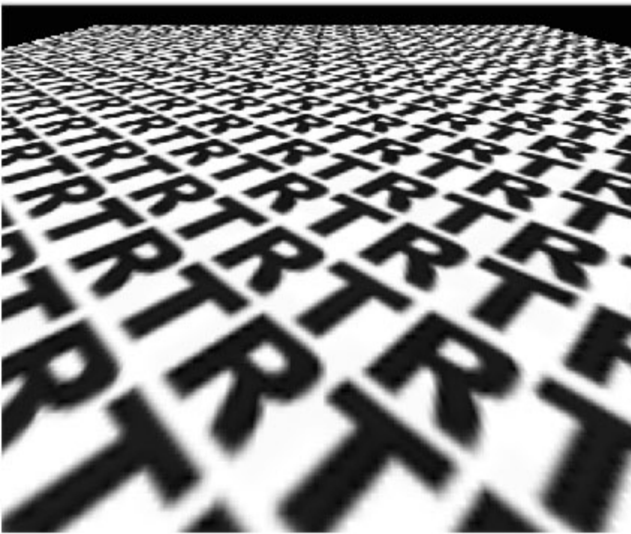
```
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);
int w, h, comp; // width, height, #components (rgb=3, rgba=4)
unsigned char* image = stbi_load("floor.jpg", &w, &h, &comp, STBI_rgb_alpha);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA, GL_UNSIGNED_BYTE, image);
free(image);
glGenerateMipmap(GL_TEXTURE_2D);

//Indicates that the active texture should be repeated over the surface
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
// Sets the type of mipmap interpolation to be used on magnifying and
// minifying the active texture. These are the nicest available options.
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, 16);
```

Do every time you want to use this texture when drawing:

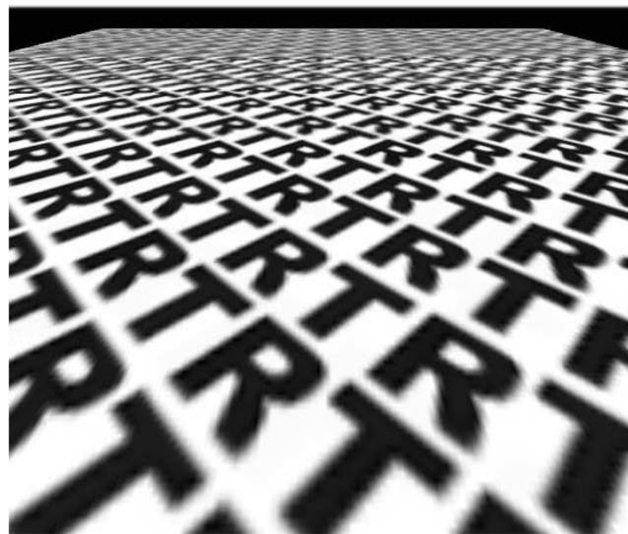
```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture);
```

Anisotropic filtering

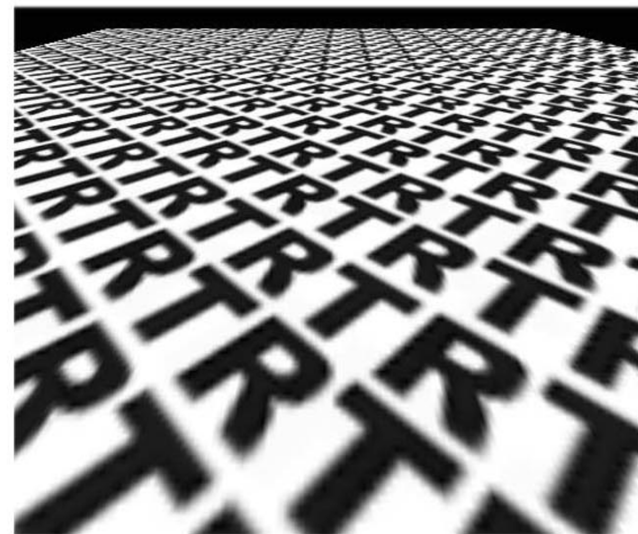




No filtering



Mipmapping



Anisotropic

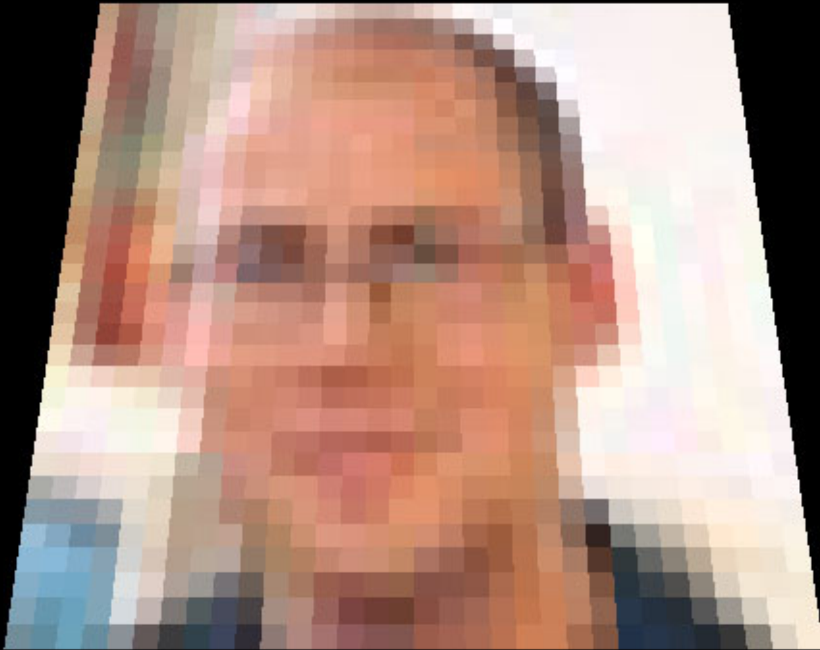
Anisotropic filtering and mipmap generation

Enabling anisotropic filtering:

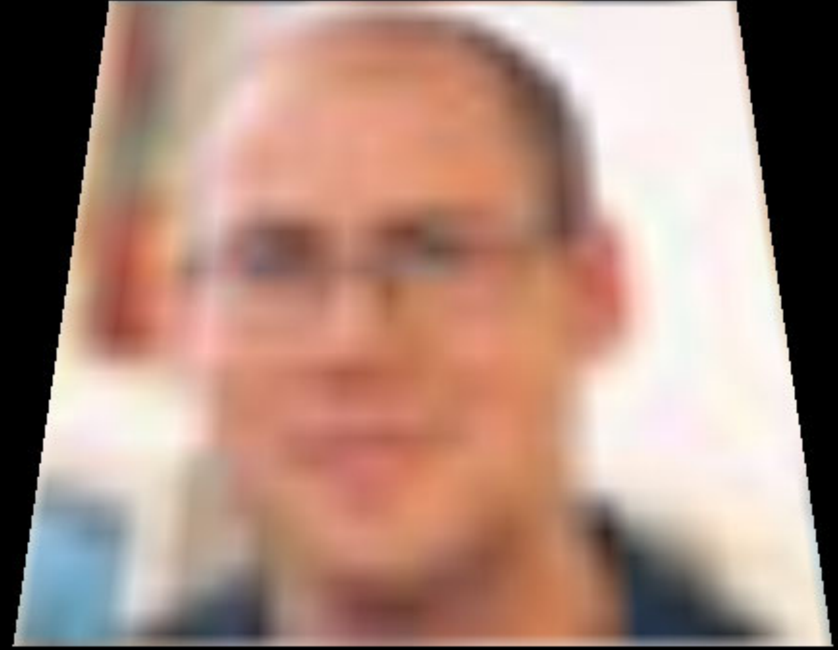
- float MaxAnisotropy
- glGetFloatv(GL_MAX_TEXTURE_MAX_ANISOTROPY_EXT, &MaxAnisotropy);
- glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, MaxAnisotropy);

- glGenerateMipmap(GL_TEXTURE_2D);

Examples of filtering



Nearest



Linear

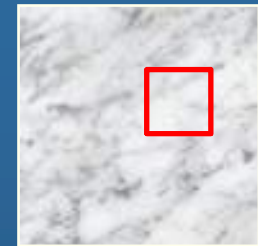
Specifying a Texture: Other Methods

- 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(...)`

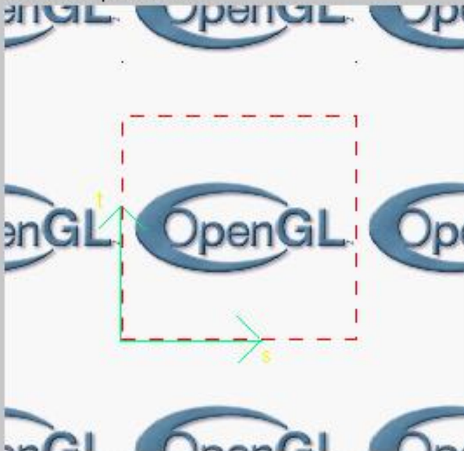


- Do both with ***`glCopyTexSubImage2D(...)`***, etc.

Example of using texturing – old way

Screen-space view

Texture-space view



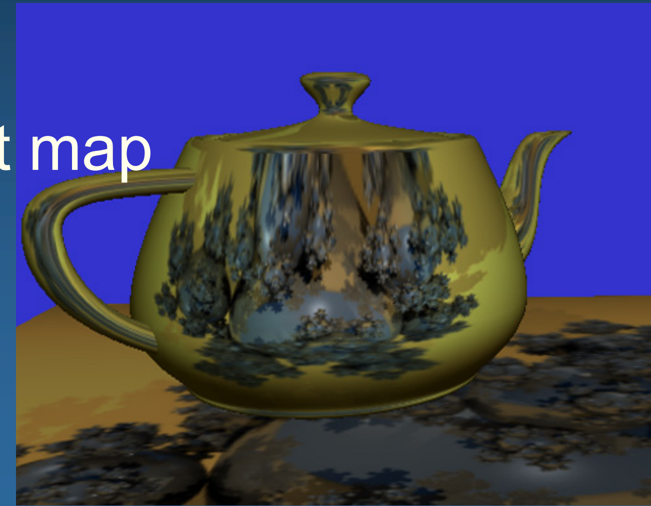
Command manipulation window

```
GLfloat border_color[ ] = {  1.00 ,  0.00 ,  0.00 ,  1.00 };
GLfloat env_color[ ] = {     0.00 ,  1.00 ,  0.00 ,  1.00 };

glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, border_color);
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, env_color);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

glEnable(GL_TEXTURE_2D);
gluBuild2DMipmaps(GL_TEXTURE_2D, 3, w, h, GL_RGB, GL_UNSIGNED_BYTE, image);
glColor4f(   0.60 , 0.60 , 0.60 , 1.00  );
glBegin(GL_POLYGON);
glTexCoord2f( 0.0 , 0.0 ); glVertex3f( -1.0, -1.0, 0.0 );
glTexCoord2f( 1.0 , 0.0 ); glVertex3f( 1.0 , -1.0, 0.0 );
glTexCoord2f( 1.0 , 1.0 ); glVertex3f( 1.0 , 1.0 , 0.0 );
glTexCoord2f( 0.0 , 1.0 ); glVertex3f( -1.0, 1.0 , 0.0 );
glEnd();
```

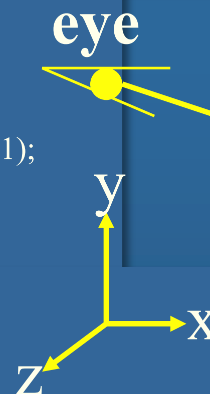
Click on the arguments and move the mouse to modify values.

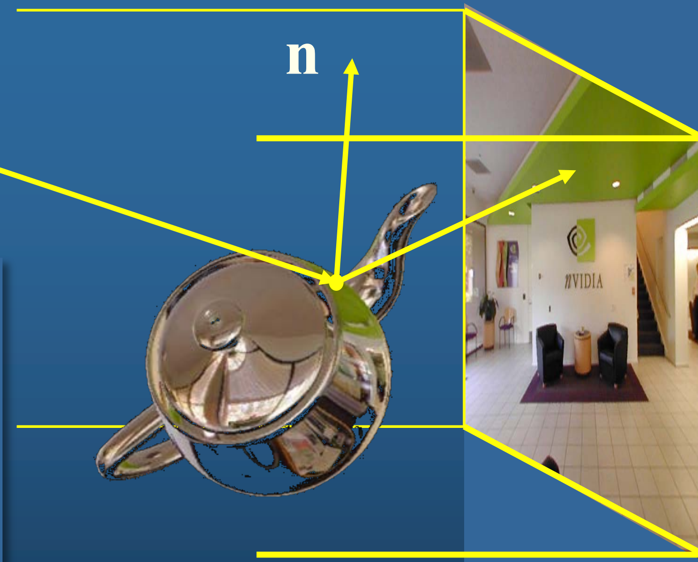
Reflections with environment mapping

- 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;
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;
}
```



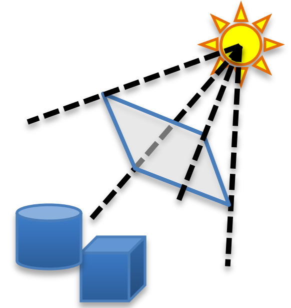



```
FRAGMENT SHADER
in vec3 normal;
in vec3 eyeVector;
uniform samplerCube tex0;
out vec4 fragmentColor;

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

Shadow Maps

Shadow Maps

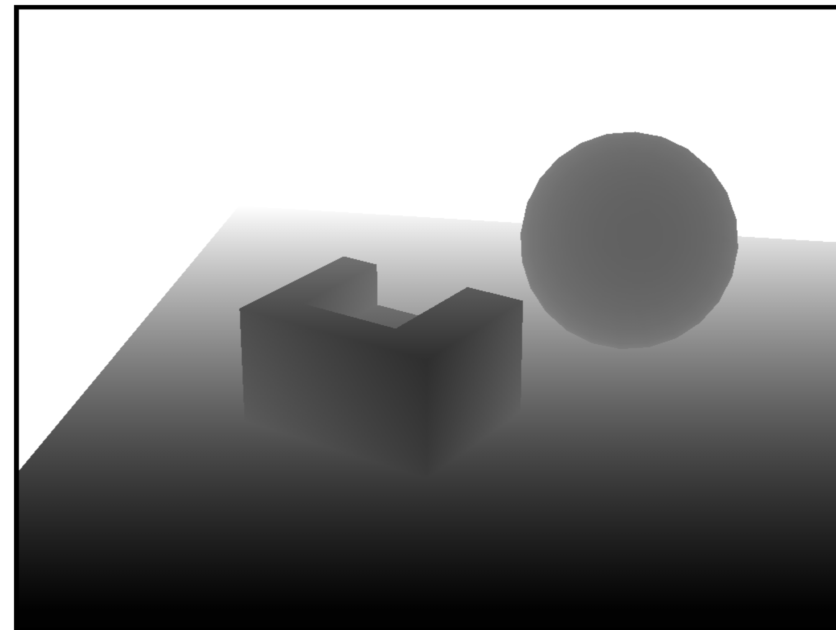
Shadow Maps



Basic Algorithm – the simple explanation:

Idea:

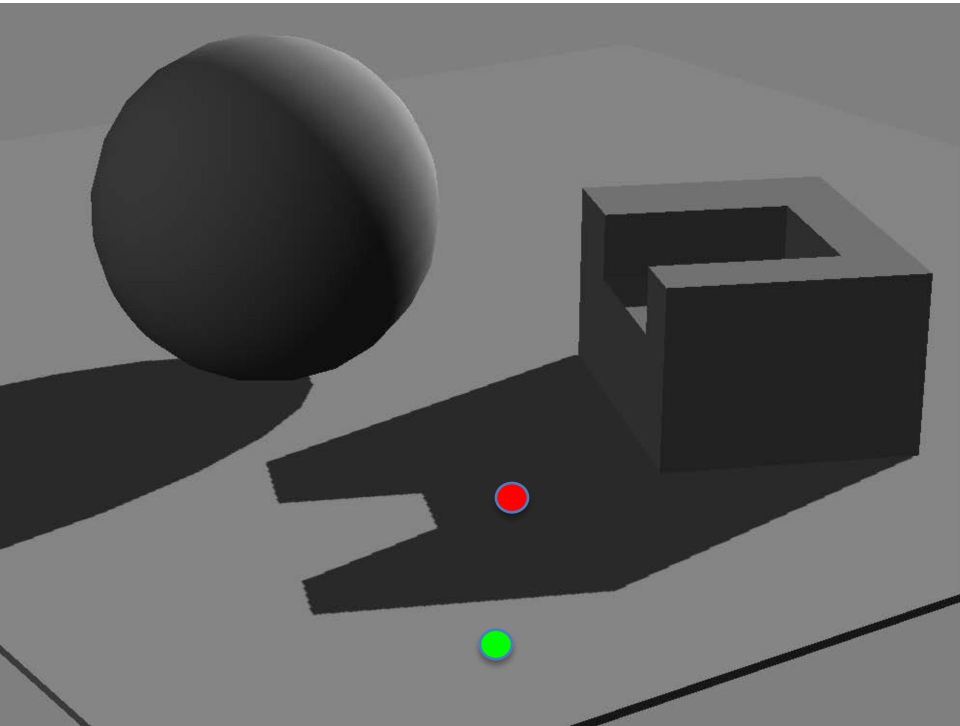
- Render image from light source
  - Represents geometry in light
- Render from camera
  - Test if rendered point is visible in the light's view
    - If so -> point in light
    - Else -> point in shadow



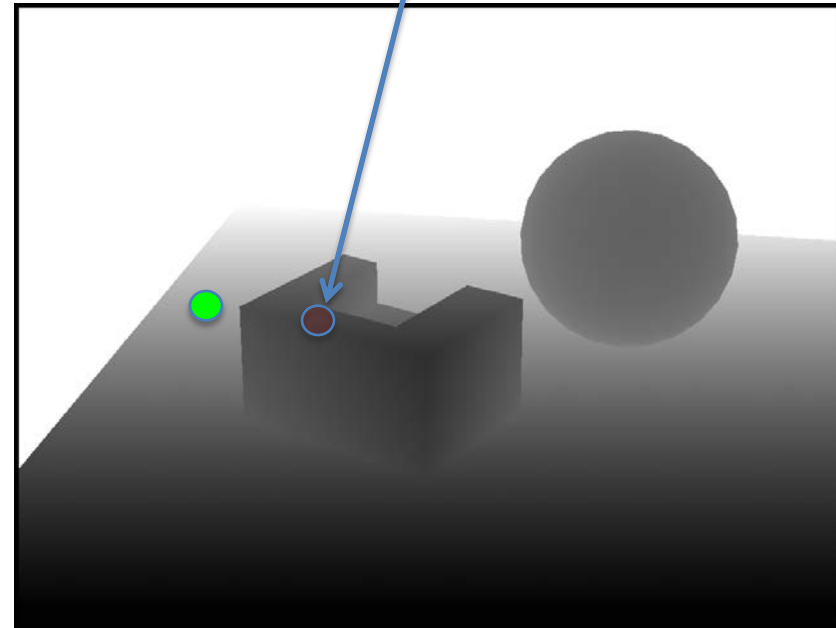
Shadow Map (light's view)

Shadow Maps

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



Camera's view



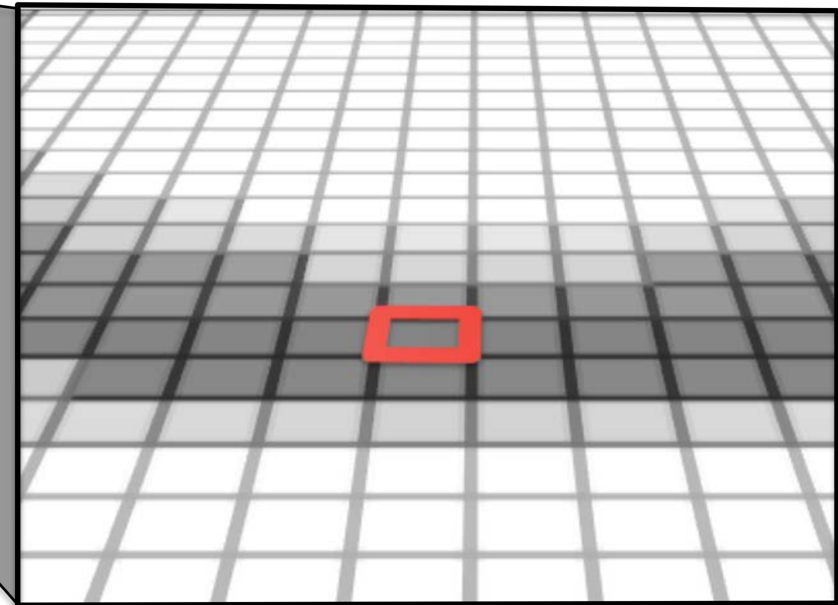
Light's view (Shadow Map)

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

Shadow Map



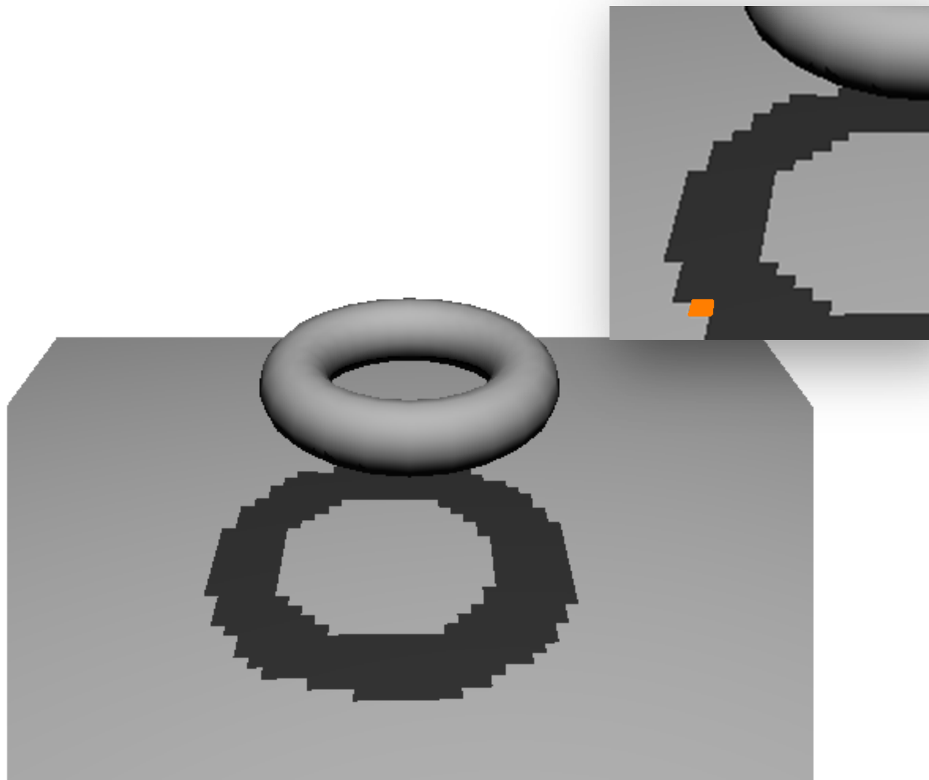
Camera's view

Shadow Maps

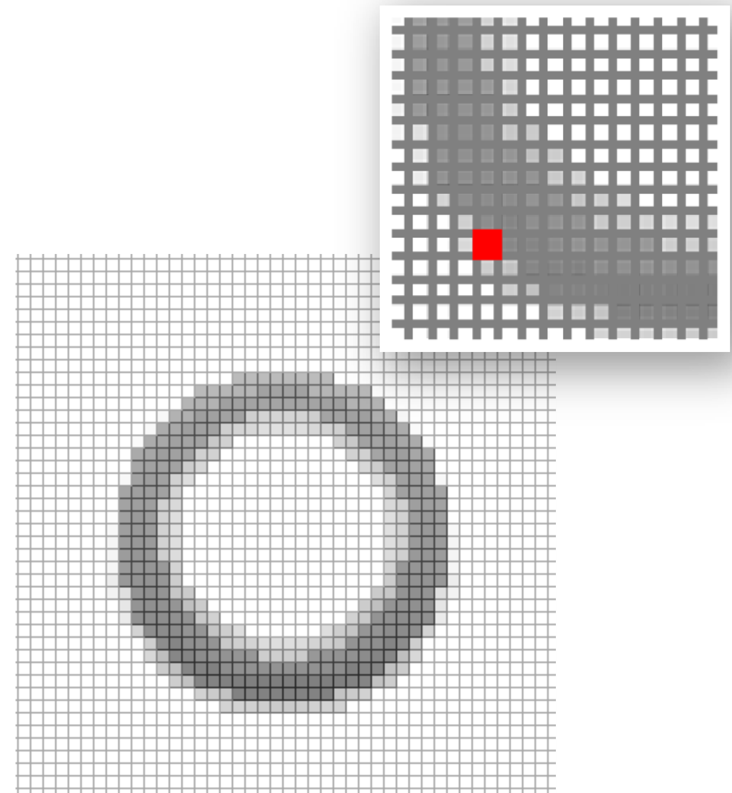
- Pros
  - Very efficient: “This is as fast as it gets”
- Cons…

Shadow Maps - Problems

- Low Shadow Map resolution results in jagged shadows



from viewpoint



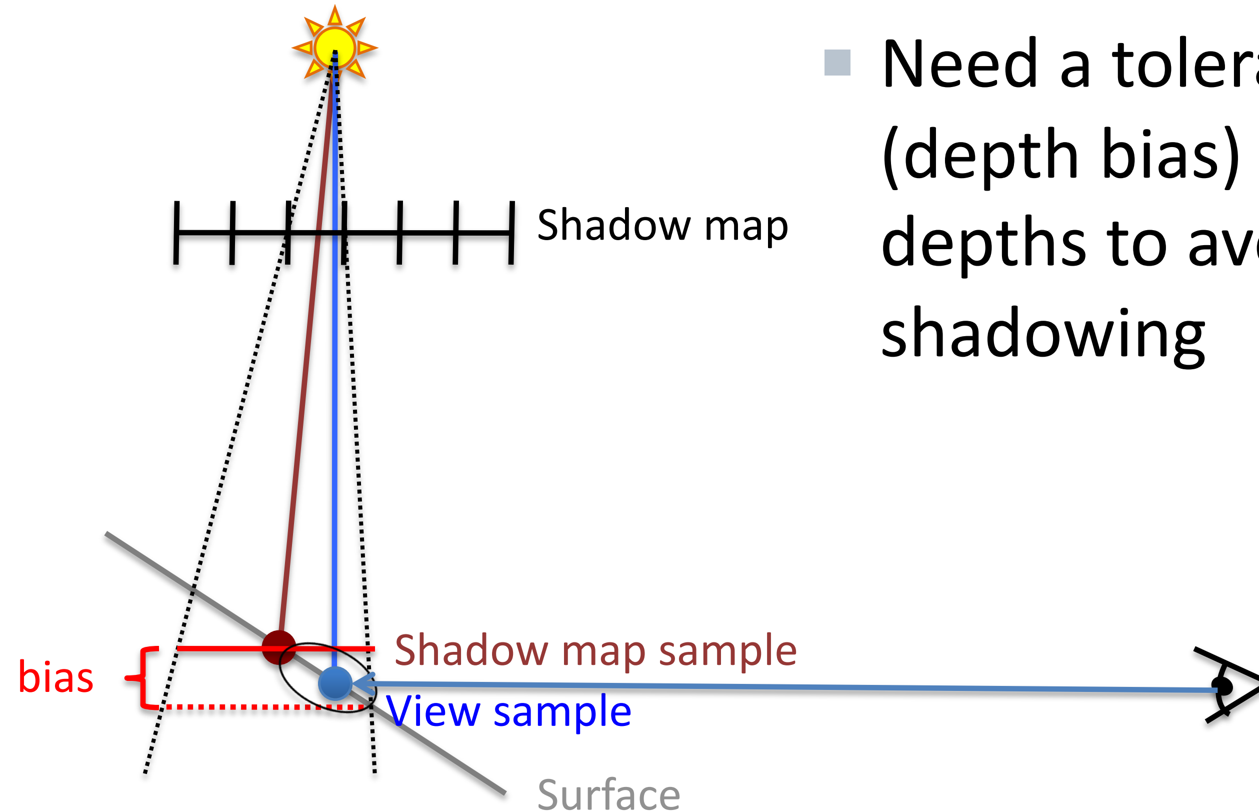
from light

Shadow Maps - Problems

In addition:

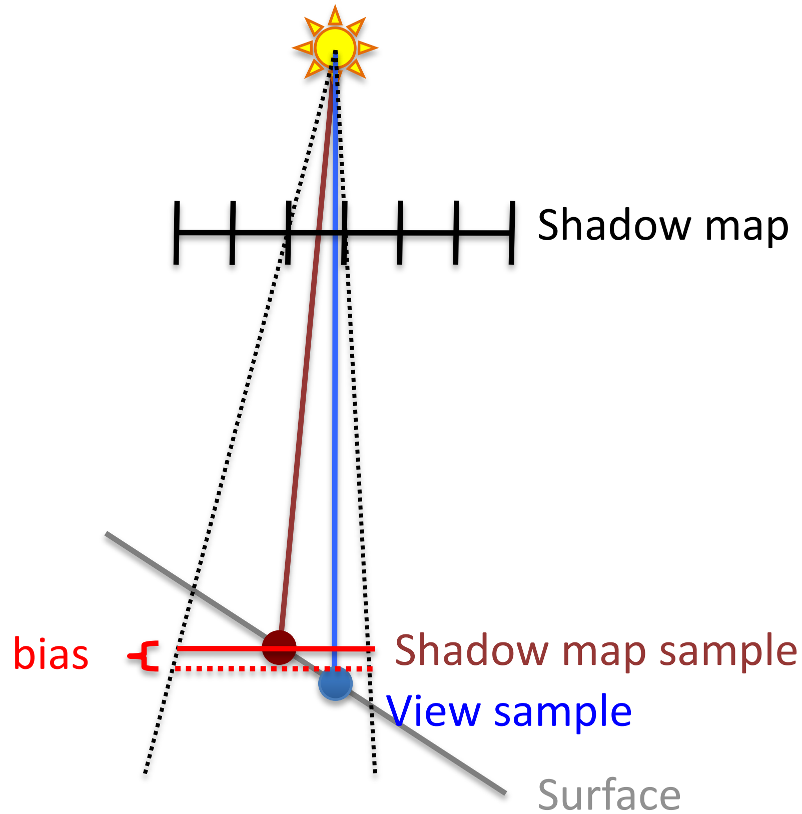
- A tolerance threshold (bias) needs to be tuned for each scene for the depth comparison

Bias

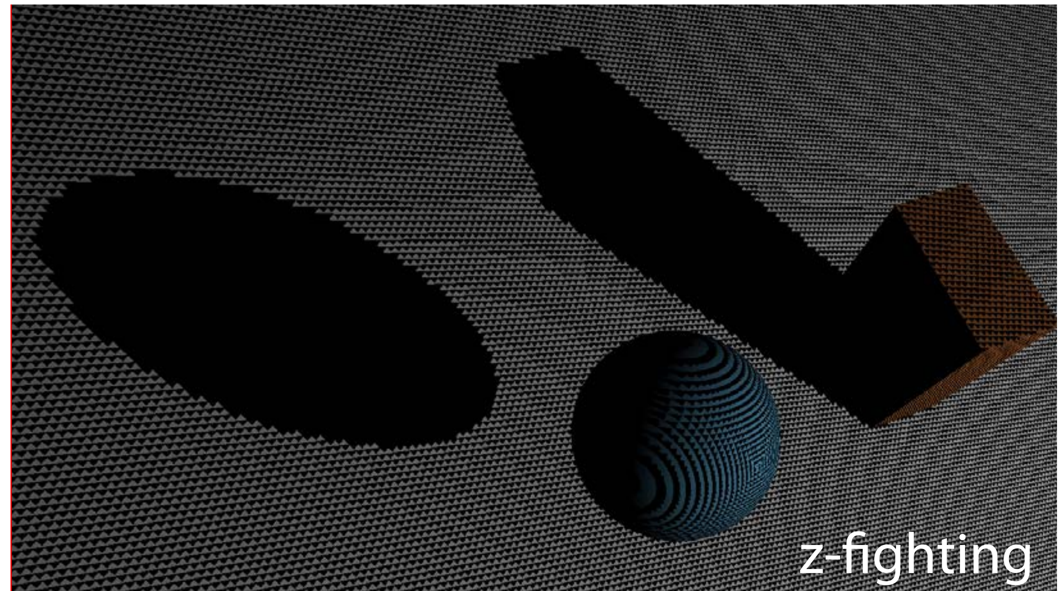


- Need a tolerance threshold (depth bias) when comparing depths to avoid surface self shadowing

Bias

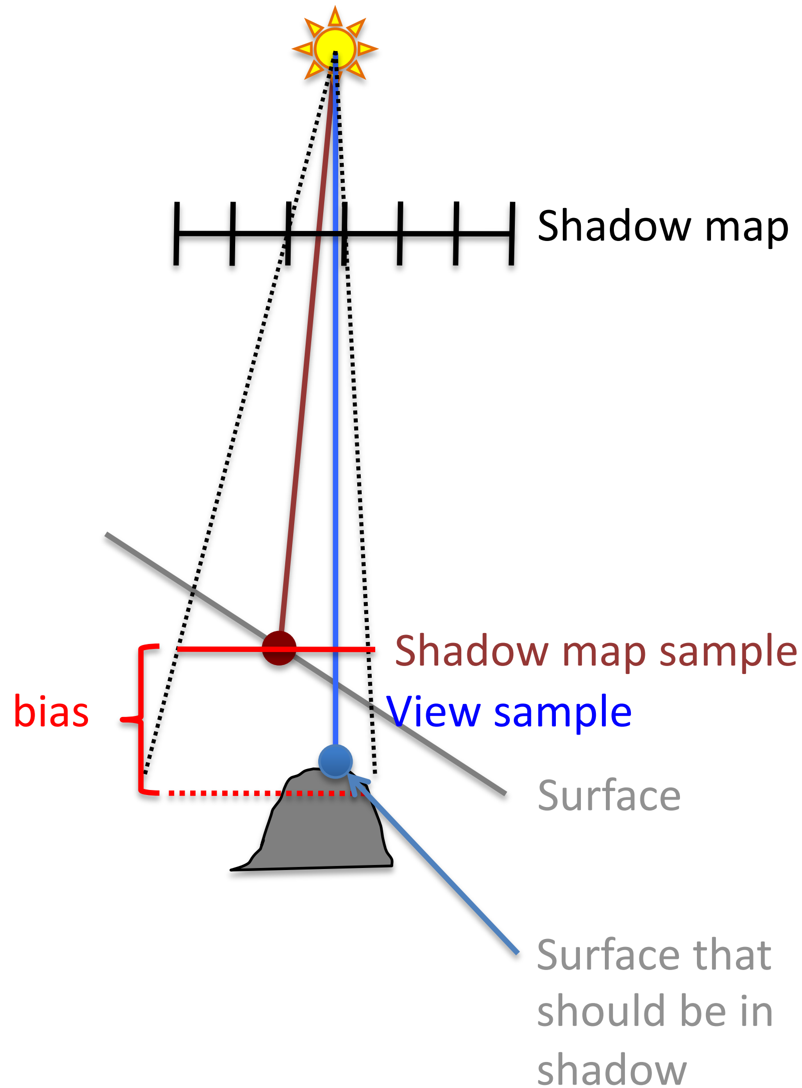


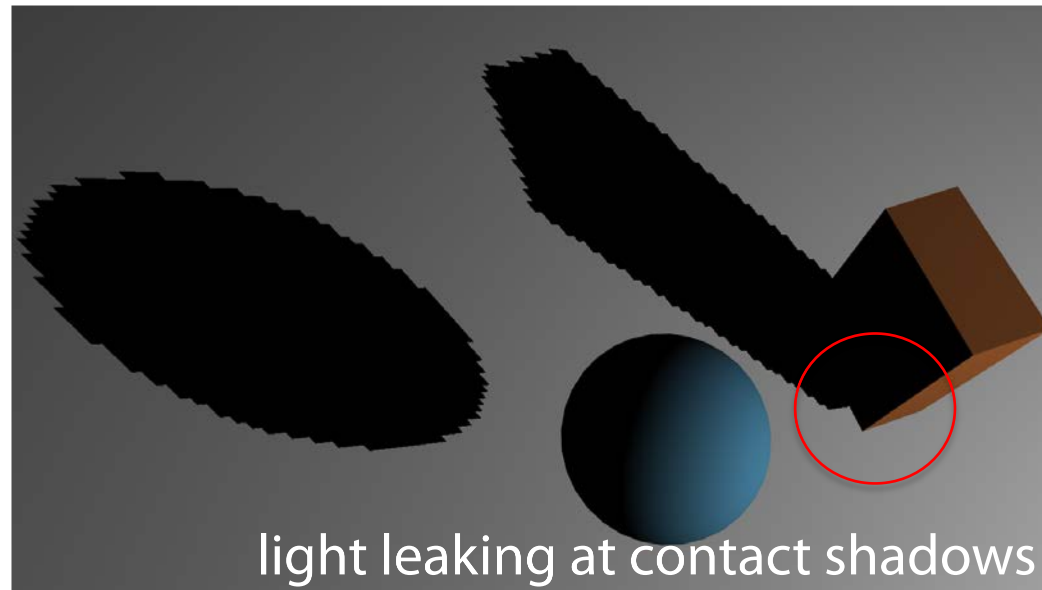
- Need a tolerance threshold (depth bias) when comparing depths to avoid surface self shadowing



Bias

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





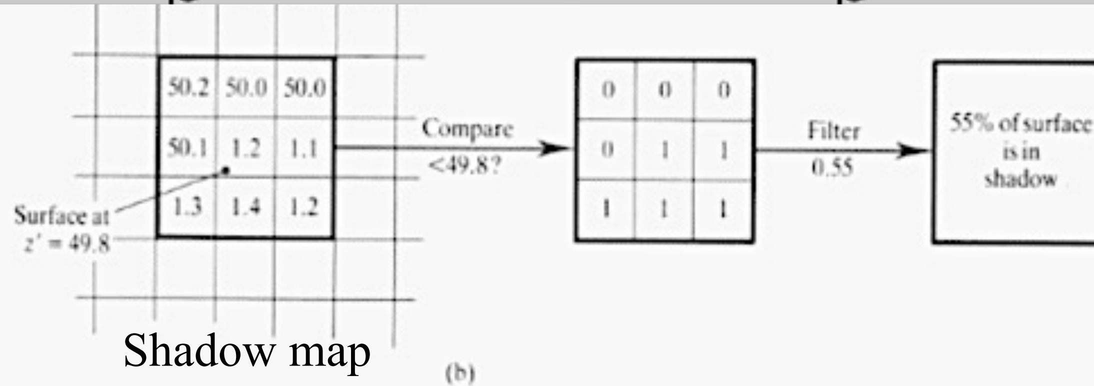
Implementing Shadow Maps

- 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

Percentage Closer Filtering



Blending

- Used for
  - Transparency
    - **glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)**
    - **glBlendEquation()**
    - **glBlendFuncSeparate() / glBlendEquationSeparate()**
  - Effects (shadows, reflections)
  - Complex materials
    - Quake3 uses up to 10 rendering passes, blending toghether contributions such as:
      - Diffuse lighting (for hard shadows)
      - Bump maps
      - Base texture
      - Specular and emissive lighting
      - Volumetric/atmospheric effects
  - Enable with **glEnable(GL_BLEND)**

Ulf Assarsson © 2003

Example of blending for Motion Blur

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

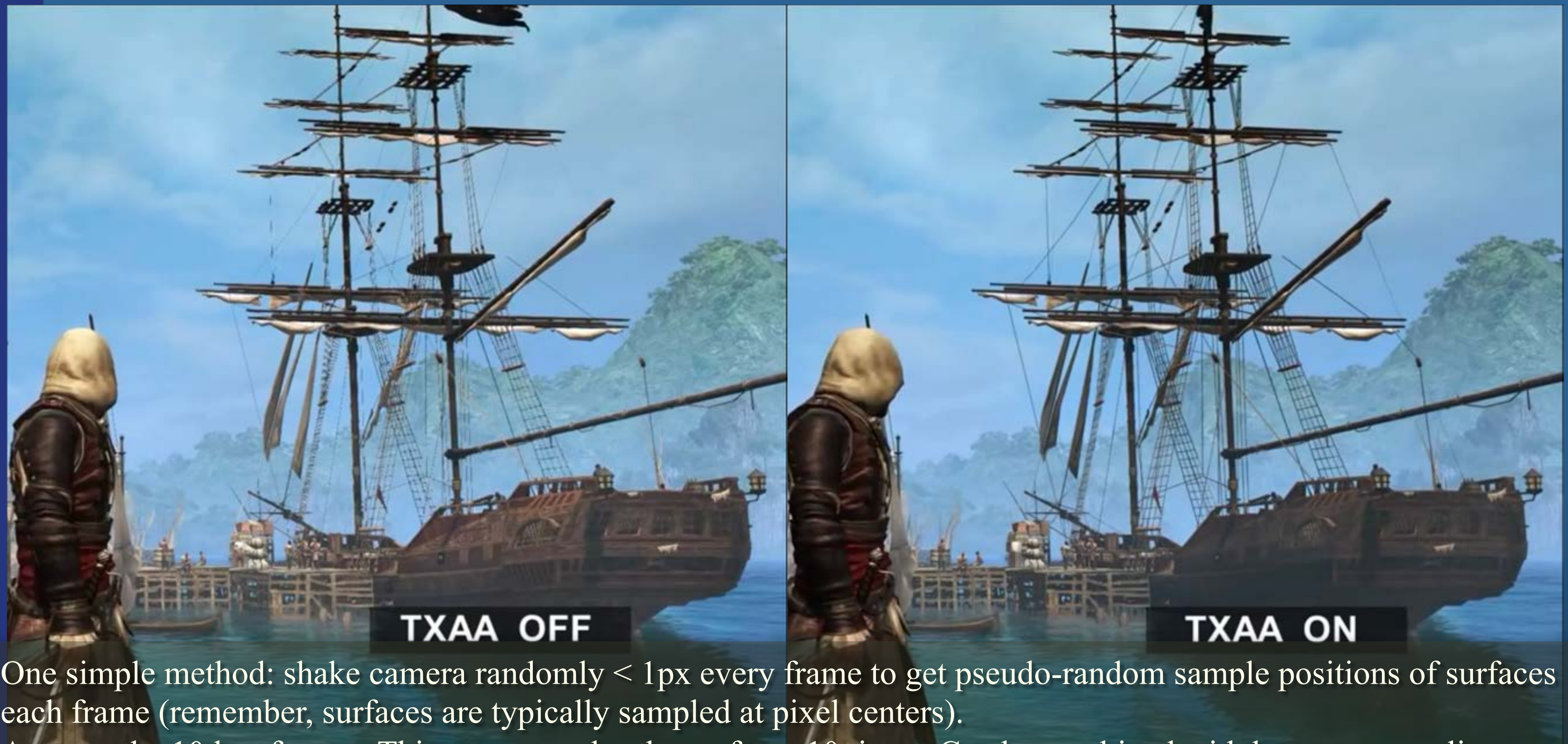
Image courtesy Brostow and Essa

Ulf Assarsson © 2003

Temporal Anti-aliasing (TXAA)

- Assasin'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 < 1px 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 -> utilize their motion vectors to fetch correct screen-space sample from frame (n-i), $i \in [0,9]$

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,
–AUX*i (where i is [0, AUX_BUFFERS - 1]), 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
  - Back/front/left/right – **glDrawBuffers()**
- Depth buffer (z-buffer)
  - For correct depth sorting
    - (instead of BSP-algorithm, painters algorithm…)
  - **glDepthFunc(),**
  - **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 stecilbuffer = 0.
    - less, lequal, greater, gequal, always
    - **glStencilFuncSeparate(…)** – settings for back-facing vs front-facing polygons
  - **glStencilOp** – how to update the stencil buffer: keep, replace, incr, decr
  - **(glStencilMask,)**
- 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()`
- ~~Fog: `glFog(), glEnable(GL_FOG);`~~

Fragment Operations

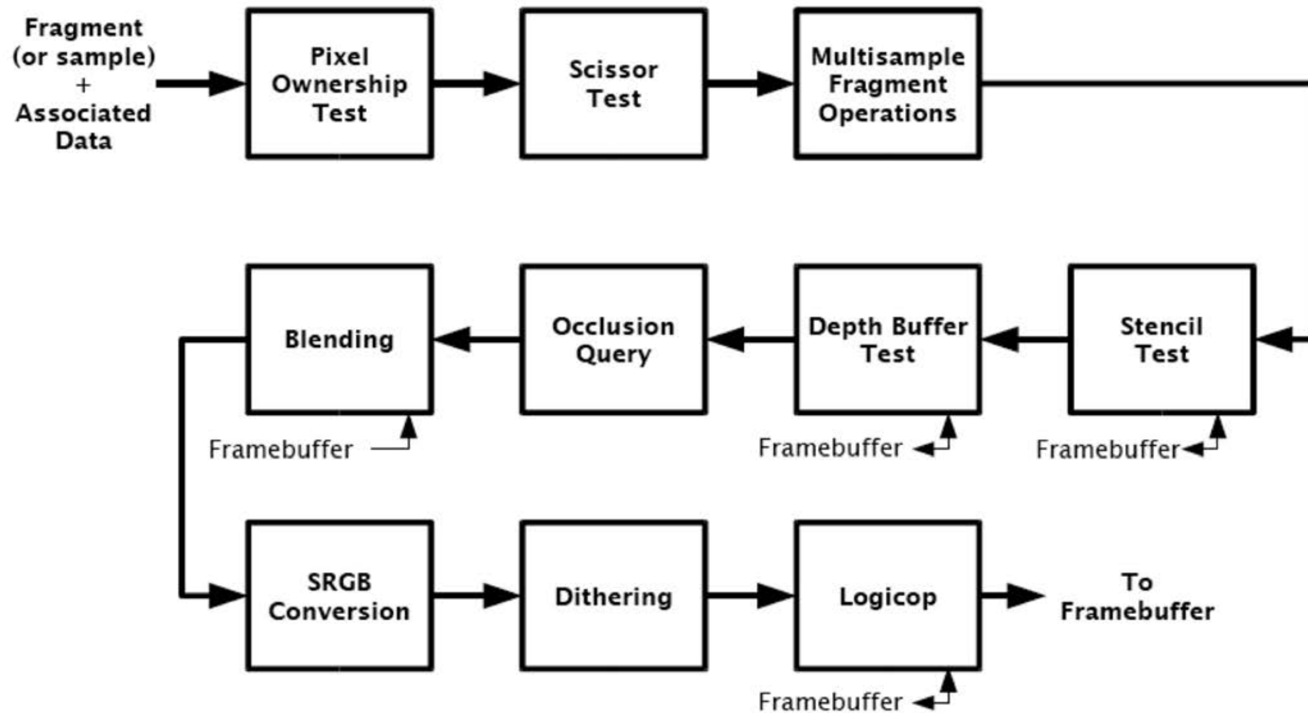


Figure 4.1. Per-fragment operations.

Extensions

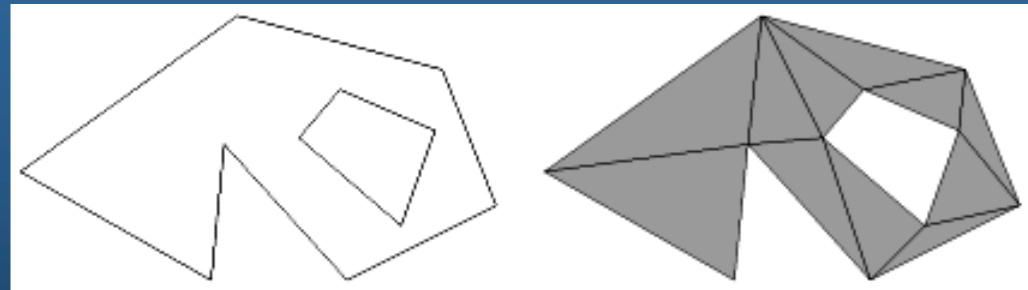
- glew.h + glew32.lib/dll OR GLee.h + GLee.cpp
- Or get the extensions manually:
- Check if extension is supported:
  glutExtensionSupported("GL_EXT_framebuffer_sRGB")
  glutExtensionSupported("GL_EXT_texture_integer")
- Get address of extension function:
  - **gTexParameterIivEXT = wglGetProcAddress(”glTexParameterIivEXT");**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

GLU – The OpenGL Graphics System Utility Library

- #include <GL/glu.h>. Loads: glu32.dll or link with glu32.lib
- Support for creating Mip maps
- Matrix manipulation functions (=camera helper functions)
- Polygon Tesselation
  - Creating arbitrary (non-convex) polygons
- Quadrics (2:nd order surfaces)
- NURBS

GLU - Polygon Tesselation

- The GLU Tesselation Functions
  1. gluTessBeginPolygon() begins a new polygon.
  2. gluTessBeginContour() begins a new contour.
  3. gluTessVertex() is called repeatedly to pass the vertices to the tesselator.
  4. gluTessEndContour() ends the contour. If there are more contours in the polygon, continue at Step 2.
  5. gluTessEndPolygon()



A concave polygon with one hole (left) and the same polygon after tesselation (right)

GLU - Quadrics

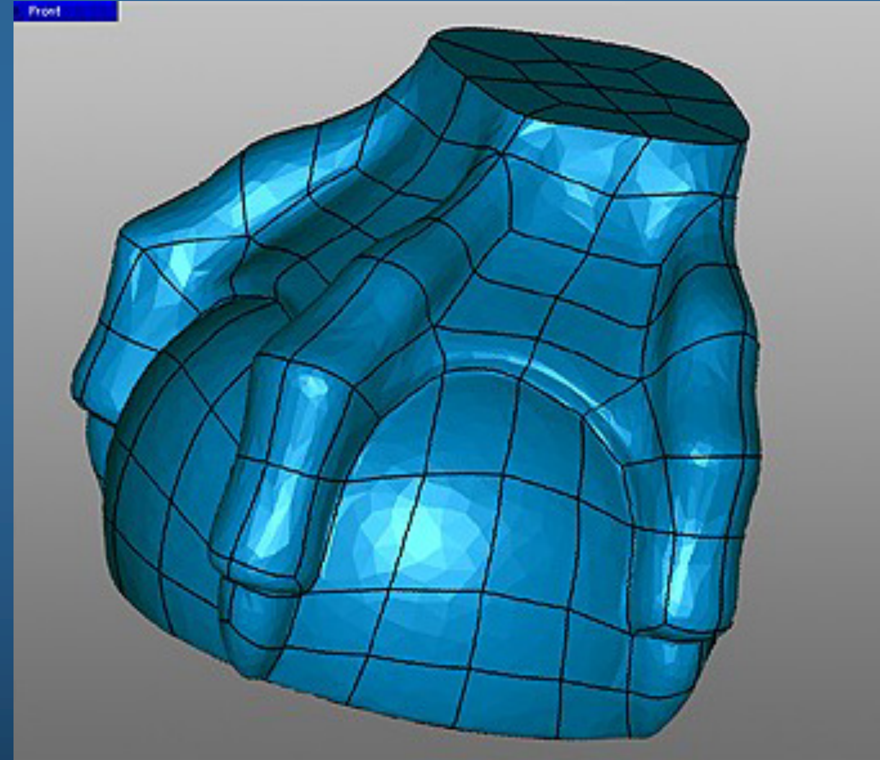
- To render spheres, cylinders and disks.
  - Example:

```
GLUquadricObj *gQuad;
gQuad=gluNewQuadric();
gluQuadricDrawStyle(gQuad, GLU_FILL);
gluSphere(gQuad,radius, 40,40); // slides, stacks – Draws the sphere
```

  - **gluQuadricNormals()** – **GLU_NONE, GLU_FLAT, GLU_SMOOTH**
  - **gluQuadricTexture()** – **GL_TRUE, GL_FALSE**
  - **gluQuadricOrientation()** – **GLU_OUTSIDE, GLU_INSIDE**
  - **gluQuadricDrawStyle()** – **GLU_FILL, GLU_LINE, GLU_POINT, GLU_SILHOUETTE**
- **gluSphere(), gluDisk(), gluCylinder()**

GLU - NURBS

- See chapter 7 in http://www.ce.chalmers.se/staff/uffe/glu1.3.pdf for more information.
- And chapter 24, page 34-38 in ”Introduktion till OpenGL” at course home-page



http://www.cse.chalmers.se/edu/course/TDA361/OPENGL_2006.pdf

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?

Ulf Assarsson © 2003