OpenGL

- a quick guide

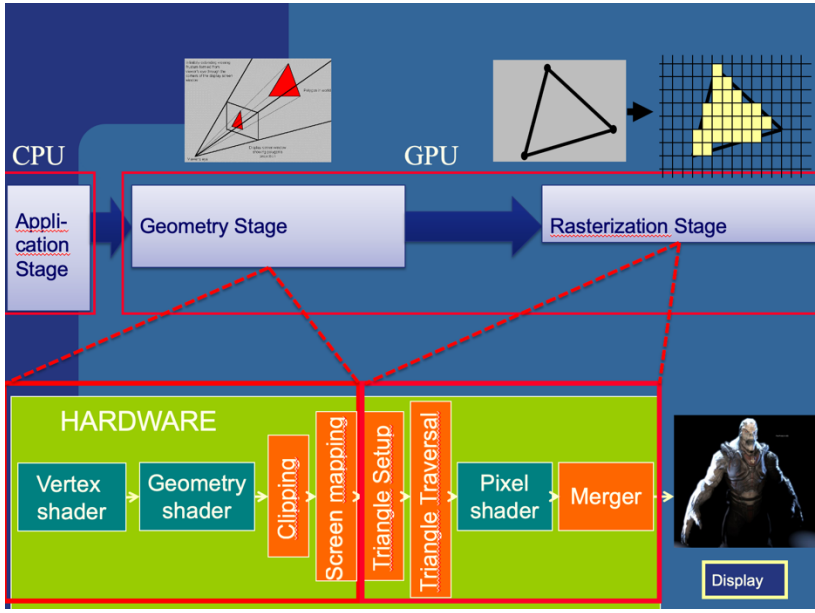
Ulf Assarsson
Department of Computer Engineering
Chalmers University of Technology

A quick summary of all previous lectures

Lecture 1

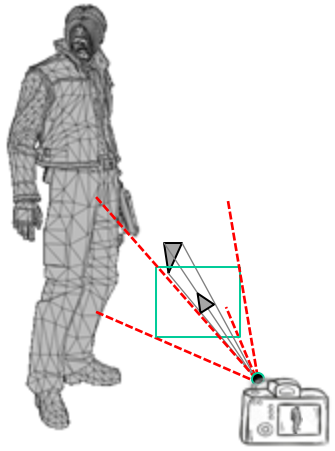
“How we do real-time 3D graphics”

- Real-time Graphics pipeline
  - Application-, geometry-, rasterization stage
- Coordinate Spaces:
  - Modelspace
    => worldspace
    => viewspace
    => homogeneous coordinates / clip space
    => normalized device coordinates
    => screen space
- Z-buffer
- Double buffering
  - Screen tearing



Lecture 2: Transforms

"How to handle cameras and motions"



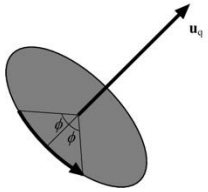
- Transformation pipeline: ModelViewProjection matrix
- Scaling, rotations, translations, projection
- Cannot use same matrix to transform normals

$$\text{Use}: \mathbf{N} = \left(\mathbf{M}^{-1}\right)^{T} \quad \text{instead of } \mathbf{M}$$

$(M^{-1})^T=M$ if rigid-body transform

- Homogeneous notation: (x,y,z,w) => [perspective division] => (x/w, y/w, z/w, 1) is a point again.
- Rigid-body transform (R,T), Euler rotation (head,pitch,roll)
- Change of frames
- Quaternions $\hat{\mathbf{q}} = (\sin\phi\mathbf{u}_q, \cos\phi)$
  - Know what they are good for. Know this:

$$\hat{\mathbf{q}}\hat{\mathbf{p}}\hat{\mathbf{q}}^{-1}$$



  - …represents a rotation of 2ϕ radians around axis $\mathbf{u}_q$ of point **p**

- Bresenhams line-drawing algorithm uses only integers (not floats)

Lecture 3.1: Shading
"How to shade your fragments"

- The triangle fragment's *color* is its *radiance*, L_{Out}, and can be split into:

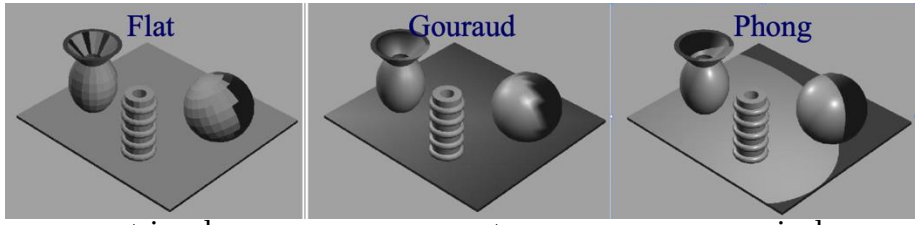
Ambient contribution
Assume homogeneous background light everywhere
$L_o = \mathbf{m}_{r,g,b}\ \mathbf{l}_{rgb}$,
where $\mathbf{m}_{r,g,b}$ is surface color and $\mathbf{l}_{rgb}$ is background-light color.

Diffuse contribution
For material part that is rough. Scale incoming light with angle to surface normal:
$L_o \mathrel{+}= \mathbf{m}_{r,g,b}\ \mathbf{l}_{rgb}\ (\boldsymbol{n} \cdot \boldsymbol{l})$,
where $\mathbf{l}_{rgb}$ is light-source color and $\boldsymbol{l}$ is light direction.

Specular contribution
For material part that is glossy (=semi-specular): Scale light-source reflection with view angle from its main reflection direction:
Phong: $L_o \mathrel{+}= \mathbf{m}_{r,g,b}\ \mathbf{l}_{rgb}\ (\boldsymbol{r} \cdot \boldsymbol{v})^{\text{shi}}$, where $\boldsymbol{r}$ is refl. direction, $\boldsymbol{v}$ = view dir.
Blinn: $L_o \mathrel{+}= \mathbf{m}_{r,g,b}\ \mathbf{l}_{rgb}\ (\boldsymbol{h} \cdot \boldsymbol{l})^{4\text{shi}}$, where $\boldsymbol{h}$ is half vector of $\boldsymbol{l}$ and $\boldsymbol{v}$.
Bonus: PBS (Physically-based Shading) - see slides

Emission contribution
Self-glowing material
$L_o \mathrel{+}= \mathbf{m}_{r,g,b}\ \mathbf{l}_{rgb}$

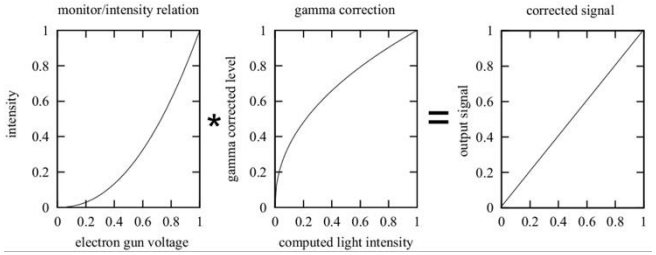
- Flat, Gouraud, and Phong shading



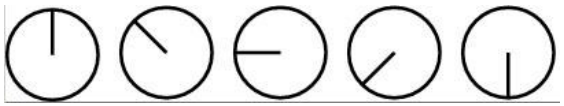
Full shading done: per triangle per vertex per pixel

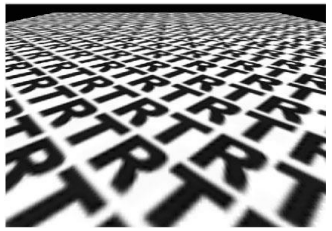
- Transparency
  Render transparent triangles in back-to-front order, with blending:
  $\mathbf{c}_o = \alpha \mathbf{c}_s + (1-\alpha)\mathbf{c}_d$
  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)

- Gamma correction

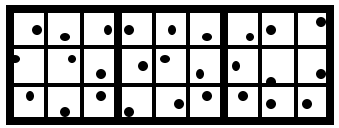


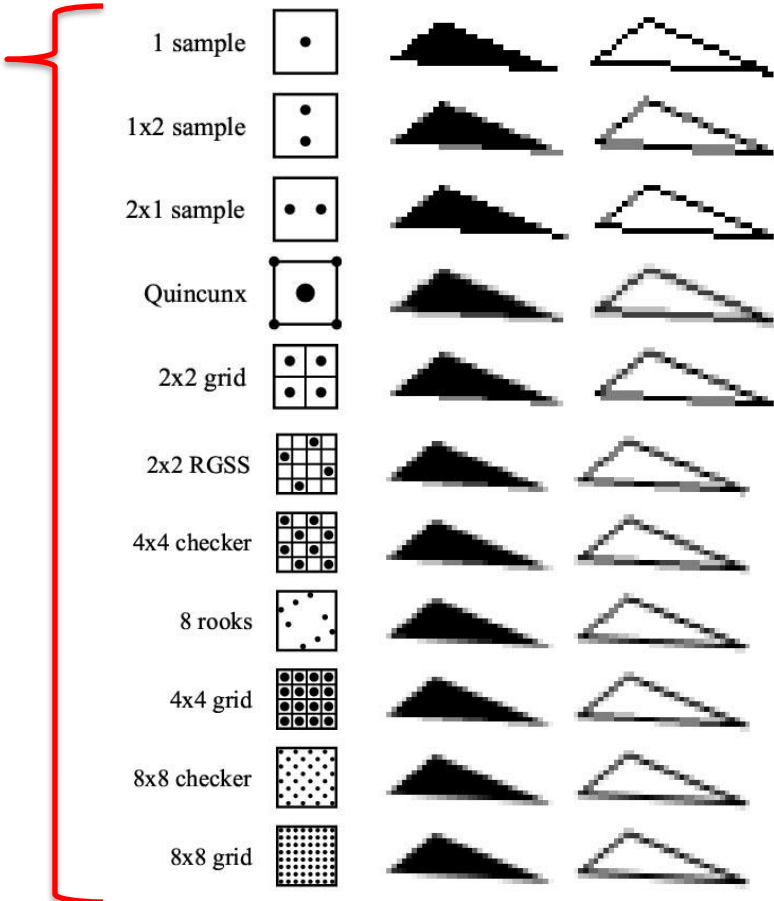
Lecture 3.2: Sampling, filtrering, and Antialiasing



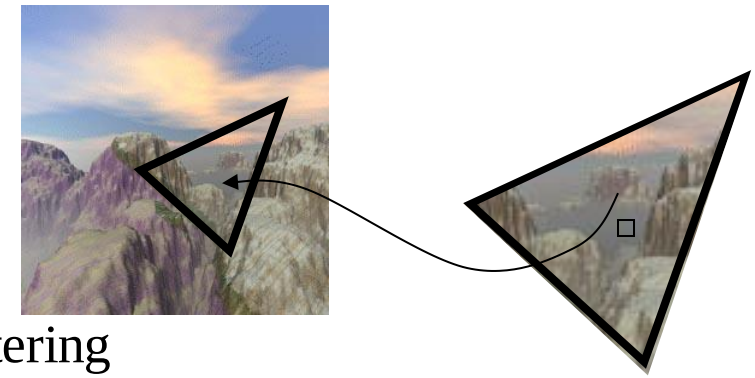


- When does aliasing occur?
  - In 1) pixels, 2) time, 3) texturing
- Supersampling schemes
- Jittered sampling
  - Why is it good?
- Supersampling vs multisampling vs coverage sampling





04. Texturing



What is most important:

- Filtering: magnification, minification
  - Mipmaps + their memory cost
  - How compute bilinear/trilinear filtering
  - Number of texel accesses for trilinear filtering
  - Anisotropic filtering
- Environment mapping – cube maps, how compute u,v from reflection vector.
- Bump mapping
- 3D-textures – what is it?
- Sprites
- Billboards/Impostors, viewplane vs viewpoint oriented, axial billboards, how to handle depth buffer for fully transparent texels.
- Particle systems

Lecture 5: OpenGL

"Overview of available functionality in graphics API:s"

- How to use OpenGL (or DirectX)
  - Will not ask about syntax. Know how to use.
    - I.e. functionality
  - E.g. how to achieve
    - Blending and transparency
    - Fog – how would you implement in a fragment shader?
      - pseudo code is enough
    - Be able to: specify a material, a triangle, how to translate or rotate an object (but only principle - not syntax).
    - Triangle – vertex order and facing

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 – cross platform
  - But stopped at around v. 4.1 for MacOS
- DirectX – for Windows only
- Vulkan – cross platform
  - But slower for MacOS (MoltenVK translates to Metal).
  - Lower level API than the others.
- Metal – MacOS only
- WebGPU – cross platform incl. web
  - But still not mature.

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. Final. Perhaps replaced by webGPU (or Vulkan)

Compatibility…

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

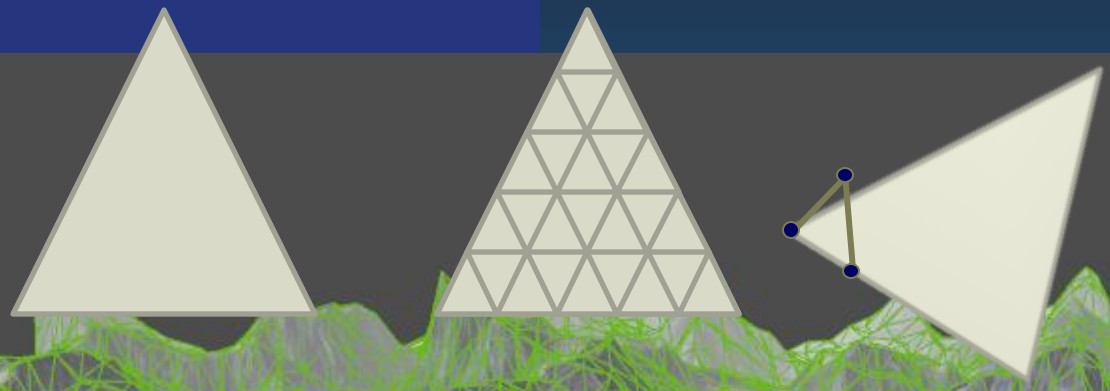
```
SDL_GL_LoadLibrary(nullptr); // Default OpenGL is fine.

// 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

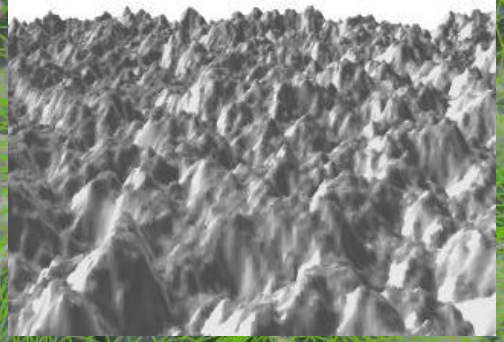
OpenGL Evolution

- Controlled by 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,**
      - **Tesselation shaders**
        - DX11: Hull shader = GL: Tesselation Control Shader
        - Domain shader = Tesselation Evaluation Shader
      - **Compute shaders (similar to OpenCL, CUDA)**
      - **Mesh shaders**
      - **Ray tracing: GLSL_NV_ray_tracing? or by mixing with OptiX, Vulcan, DirectX 12**
  - Allows for platform specific features through extensions

Tesselation – brief glance

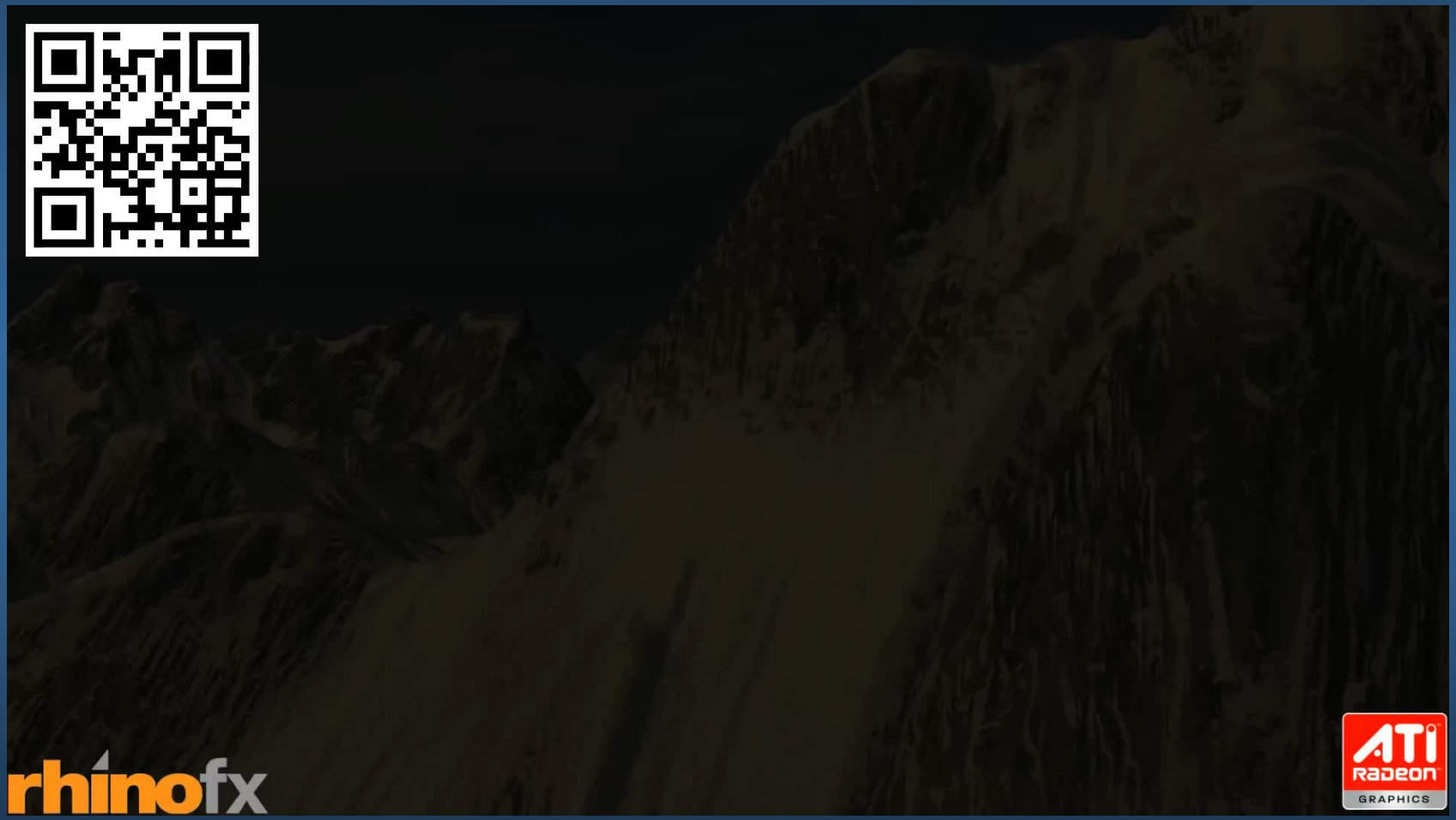


Height field



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

ATI 2007 – Ruby Whiteout



NVIDIA 2010: Endless City

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,
    - 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

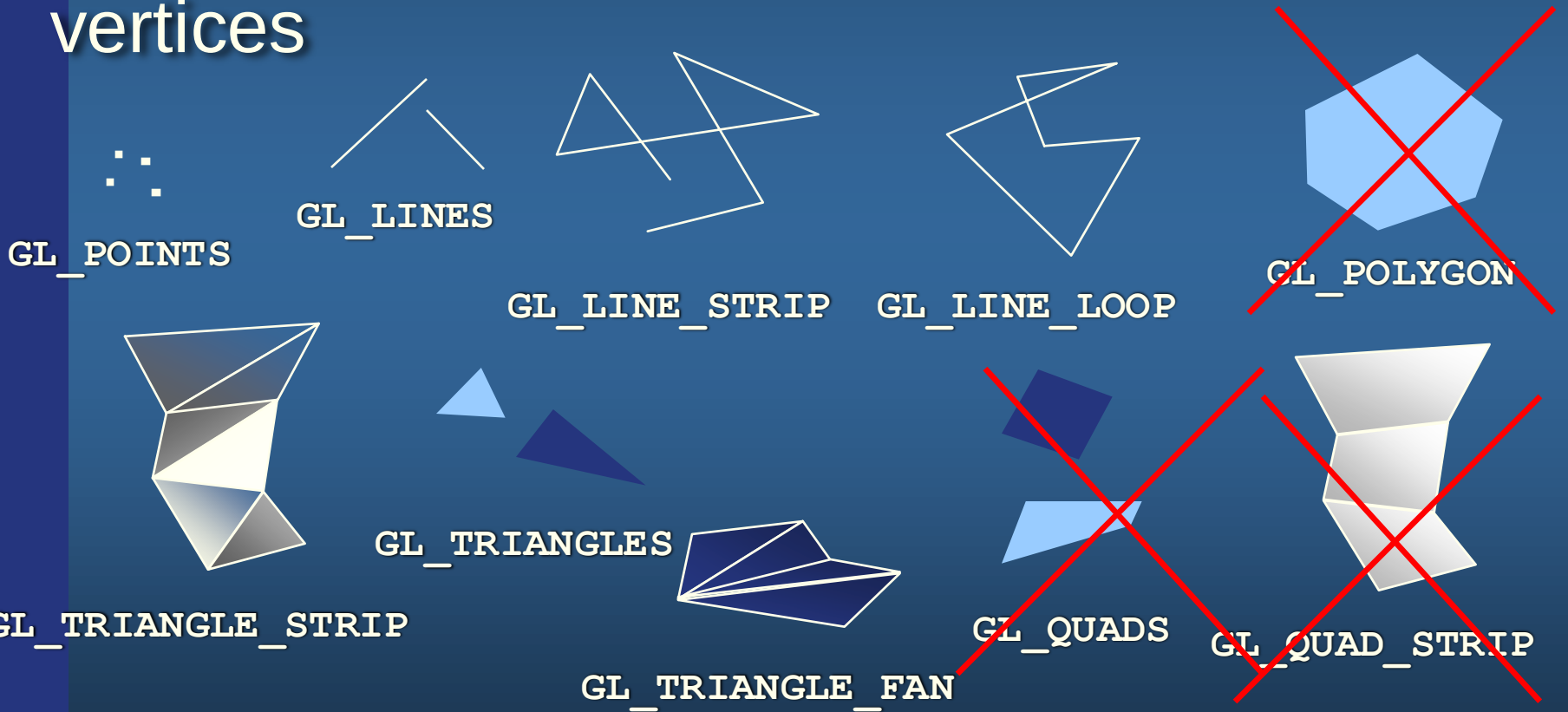
You can also find the links on the course home page:
http://www.cse.chalmers.se/edu/course/TDA362/

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)

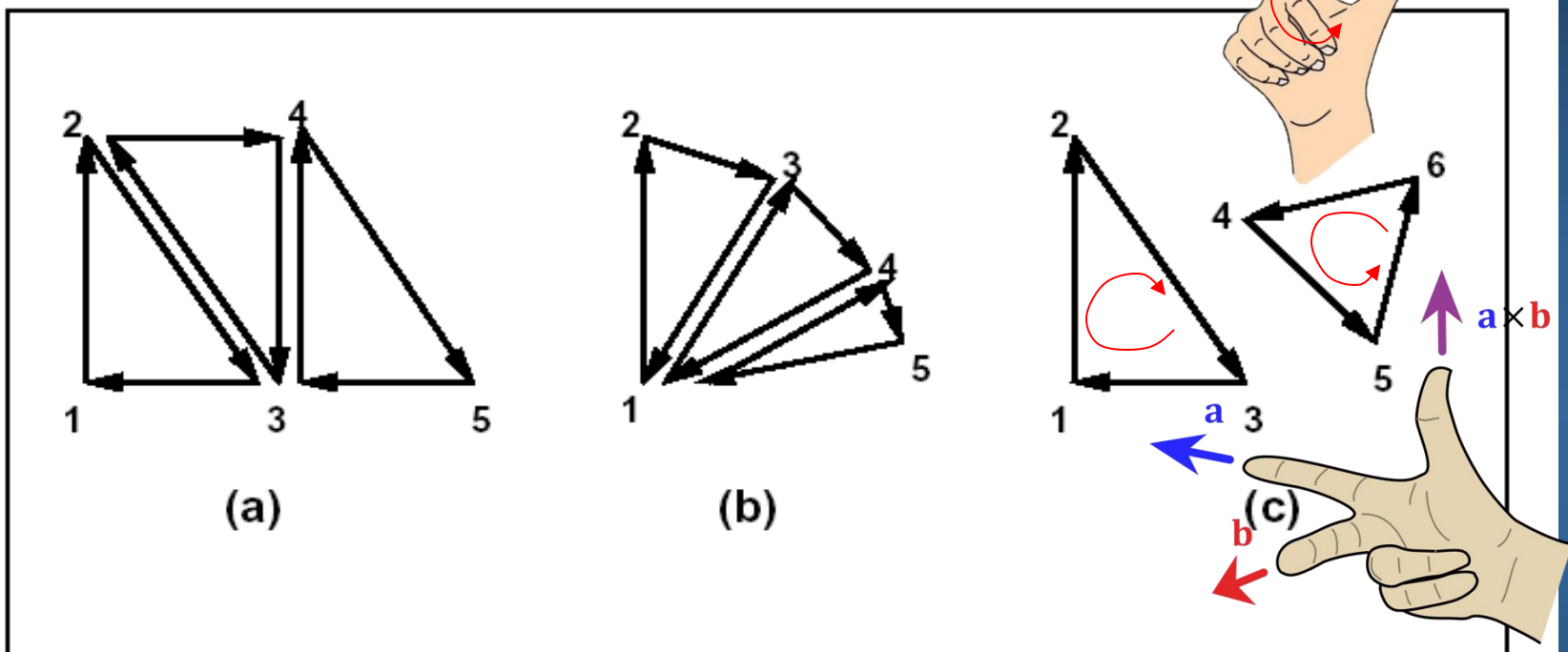


Figure 2.4. (a) A triangle strip. (b) A triangle fan. (c) Independent triangles. The numbers give the sequencing of the vertices . Note that in (a) and (b) triangle edge ordering is determined by the first triangle, while in (c) the order of each triangle's edges is independent of the other triangles.

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 USE VERTEX ARRAYS

- Vertex Arrays (Fast):

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

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

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

  void **MultiDrawElements(…)**;

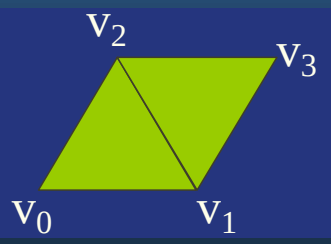
Using index list

For all options- see the OpenGL Reference Manual online

Index list:

vec3f pos[] = v_0, v_1, v_2, v_3;

uint indces[] = 0,1,2, 1,3,2;



Example of using Vertex Arrays

```
// In CPU RAM
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
}
```

1. **// SEND THE VERTEX COORDINATES TO AN OpenGL 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 );
```

```
// 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 (run on GPU)
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;
}
```

"How to send per-vertex data to GPU (positions, colors, normals, texCoords …)"

2. COMMANDS TO DRAW

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

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.:

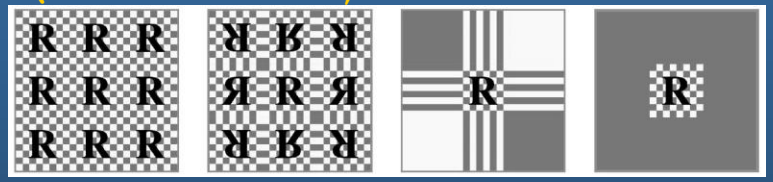
```
render(mat4 projectionMatrix, mat4 viewMatrix)
{
  glUseProgram(m_shaderProgram);
  mat4 modelViewProjectionMatrix = projectionMatrix * viewMatrix *
                                   m_modelMatrix;
  int loc = glGetUniformLocation(shaderProgram,
                "modelViewProjectionMatrix");
  glUniformMatrix4fv(loc, 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 an
glTexImage1/2/3D (), glCopyTexSubImage2D() -- set / affect ima
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

```
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
```

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

```
// 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);
}
```

glActiveTexture(0); 2. COMMANDS TO DRAW
glBindTexture(texID);
glUseProgram(shaderProgram); glBindVertexArray(vertexArrayObject); glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);

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(...)
```

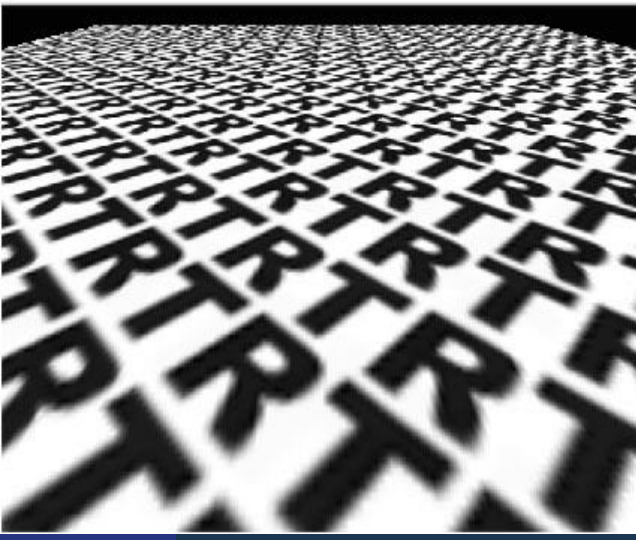
- Do both with ***glCopyTexSubImage2D(...)***, etc.

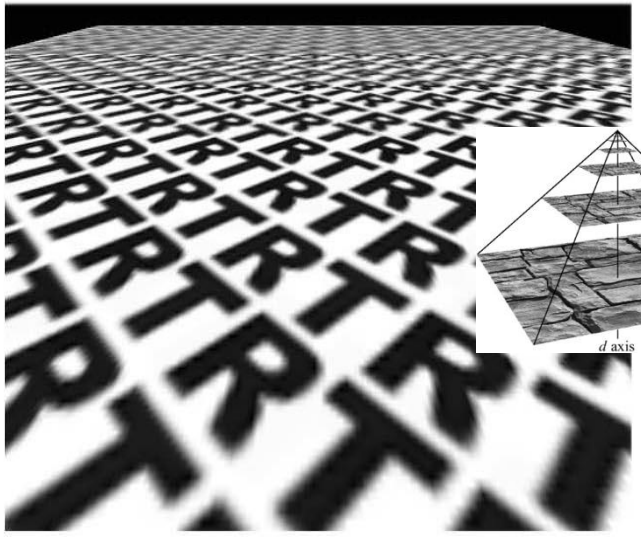
Anisotropic filtering and mipmap generation

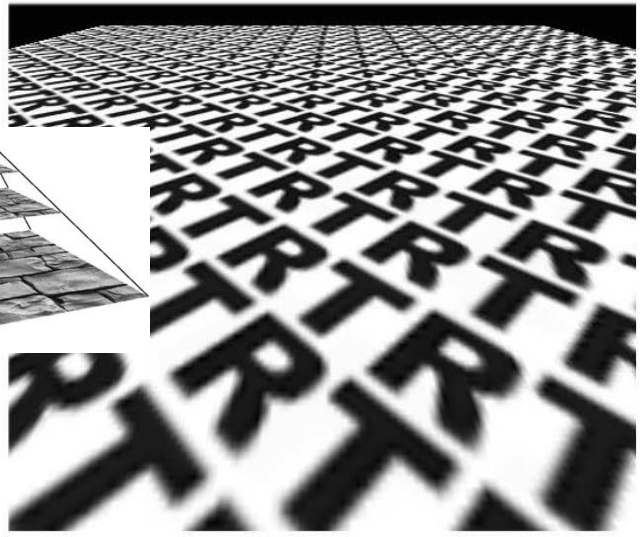
Enabling anisotropic filtering:

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

- glGenerateMipmap(GL_TEXTURE_2D);







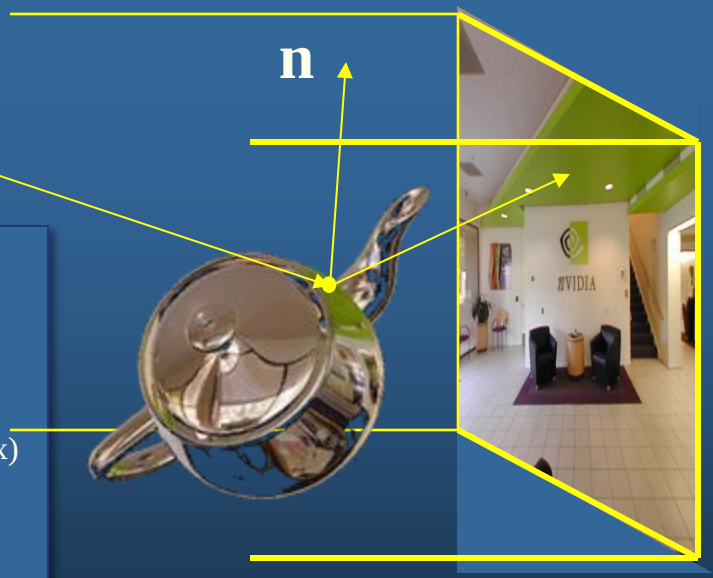
Nearest/bilinear Mipmapping Anisotropic

Reflections with environment mapping

- Texture lookups from an environment map

VERTEX SHADER

```
in vec3         vertex;         // vertex in model space
in  vec3        normalIn;       // The normal in model space
out vec3        normal;         // out: normal in view space
out vec3        eyeVector;      // out: eye vector in view space
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 view space
in vec3 eyeVector;              // in view space
uniform samplerCube tex0;
out vec4 fragmentColor;
uniform mat4 transpViewMatrix; // transpose of inverse of (inverse world–to-view-mtx)
                               // i.e., normal matrix for view-to-world transform
void main()
{
   vec3 reflectionVector = normalize(reflect(normalize(eyeVector), normalize(normal)));
   // transform reflection vector to from cam.space to worldspace before lookup…
   fragmentColor = texture(tex0, normalize(transpViewMatrix*reflectionVector));
}
```

Unity / Unreal demos 2024

UNREAL ENGINE 5.5 vs UNITY 6 | Insane Next-Gen Graphics and Best New Tech Demos

ENFANT TERRIBLE
290K subscribers

Join | Subscribe | 8K | Share | Save

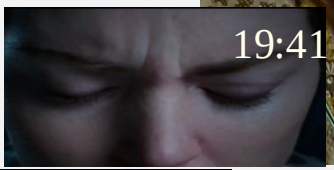
812,881 views Oct 6, 2024 #unrealengine5 #ue5 #unrealengine

Which engine would you say is the better looking at the moment? Take a look at these new TECH DEMOS from the recently announced Unreal Engine 5.5 and Unity 6 and enjoy all the INSANE GRAPHICS!

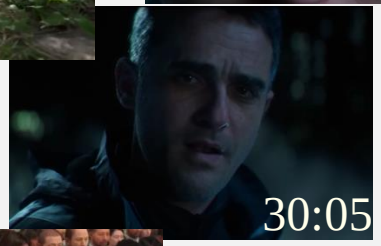
NEW ENGINES
00:00 Time Ghost (Unity 6 Tech Demo)
04:13 Unreal Engine 5.5 Tech Demo
09:23 Unreal Tournament 99 Intro (Unreal Engine 5.5)
10:19 Den of Wolves (Unity 6)

TECH DEMO
11:27 Unity Tech Demo (Book of the Dead
14:57 Unreal Engine 5 Tech Demo

REALISTIC FACE ANIMATION
16:54 Enemies Tech Demo (Unity)
18:21 OD MetaHuman (Unreal Engine 5)
19:36 Hellblade 2 MetaHuman (Unreal E
20:31 Andy Serkis MetaHuman (Unreal E



SHORT FILMS
21:56 The Heretic Short (Unity)
29:04 Blue Dot Short (Unreal Engine 5)
32:32 Lion (Unity)
32:52 The Matrix Awakens (Unreal Engine 5)



GAMES
33:55 Den of Wolves Trailer (Unity 6)
35:27 Marvel 1943: Rise of Hydra (Unreal Engine 5)
40:54 MetaHuman Tech Demo (Unreal Engine 5)

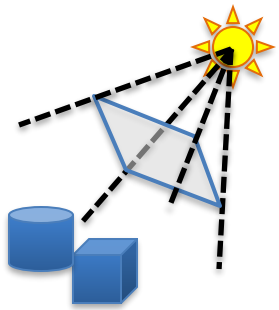
Actual games 2024/2025

https://www.youtube.com/watch?v=VJDZJnkBK7w&t=272s

Shadow Maps

Shadow Maps

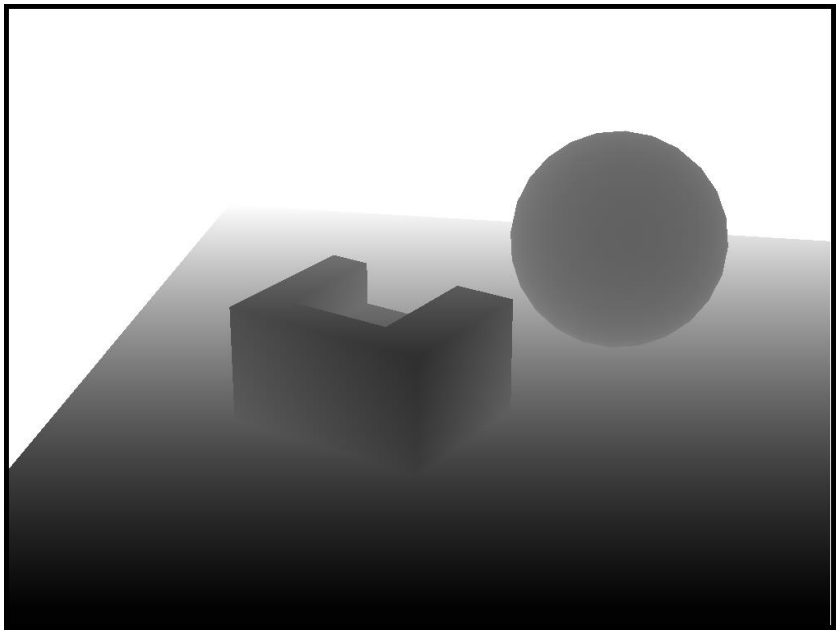
Shadow Maps



Basic Algorithm – the simple explanation:

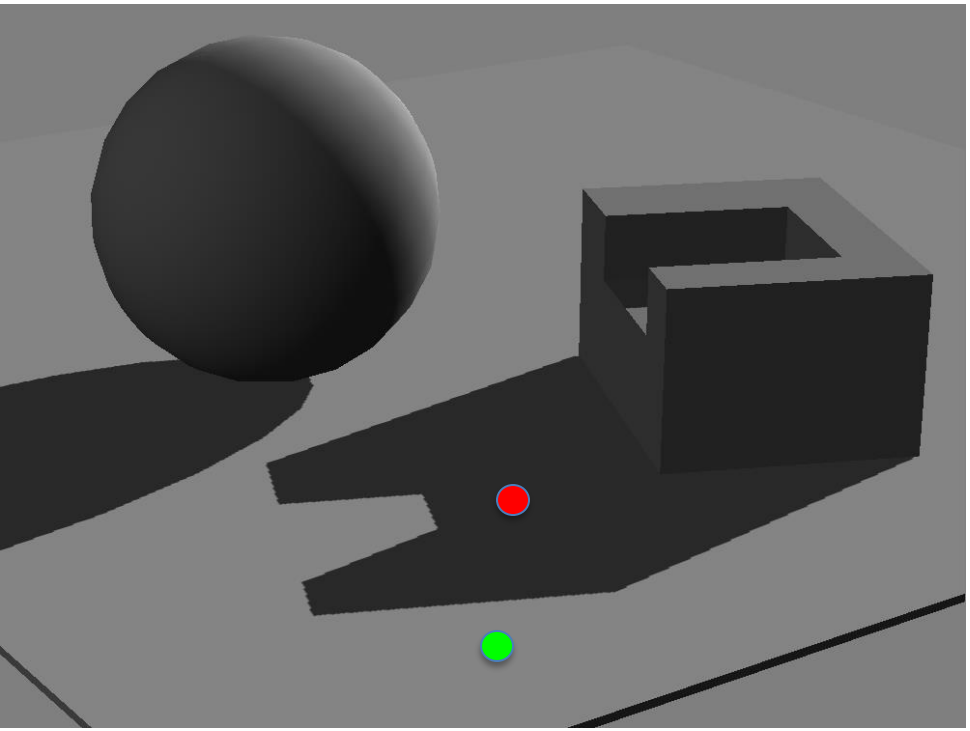
Idea:

- 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



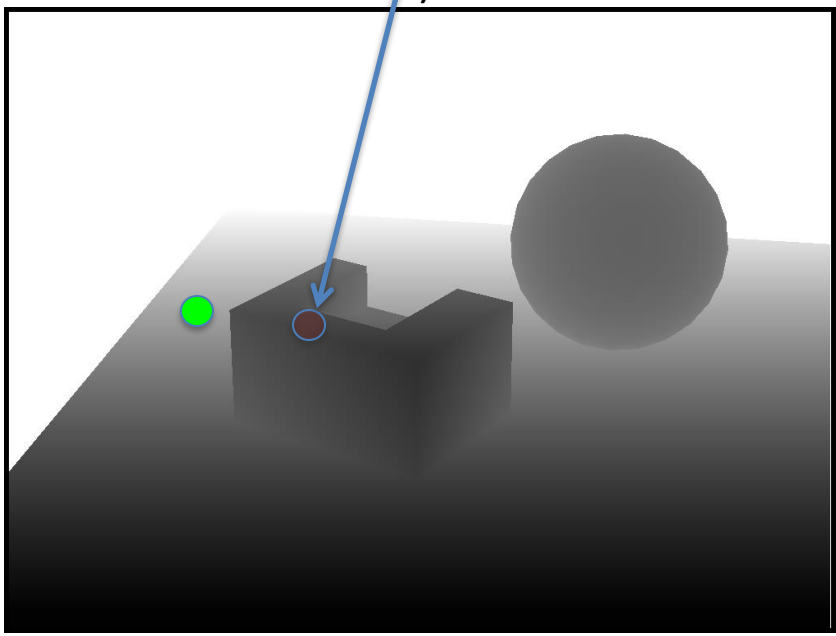
Shadow Map (light's view)

Shadow Maps



Camera's view

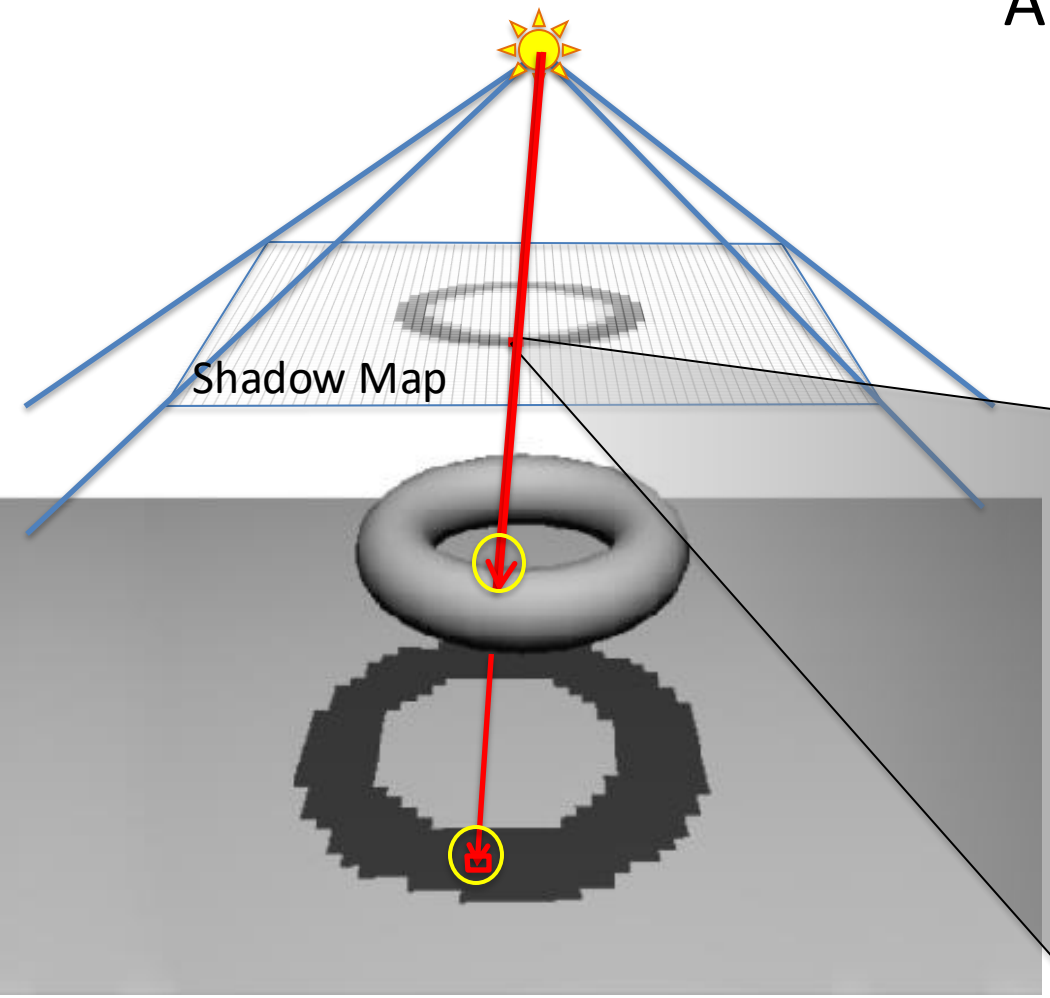
Point not represented in shadow map (point is behind box)



Light's view (Shadow Map)

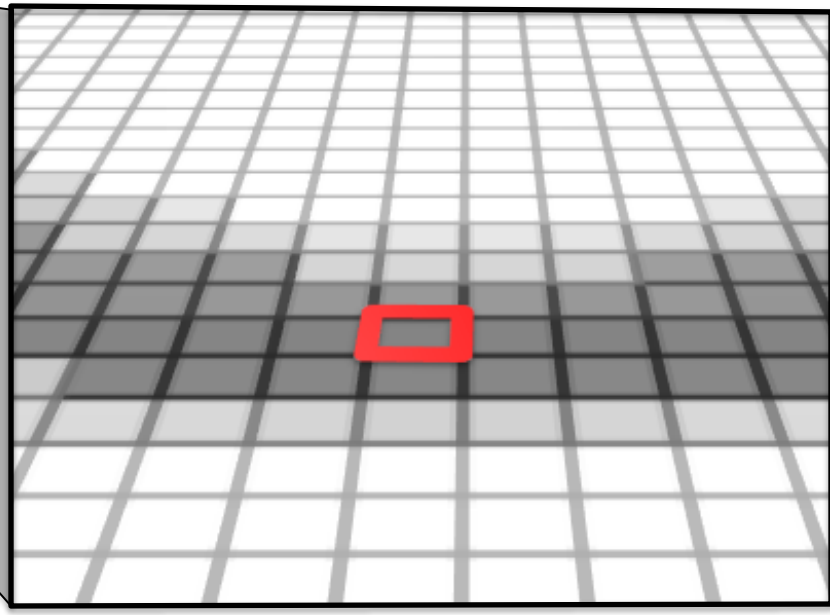
Depth Comparison

Render depth image from light



Camera's view

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

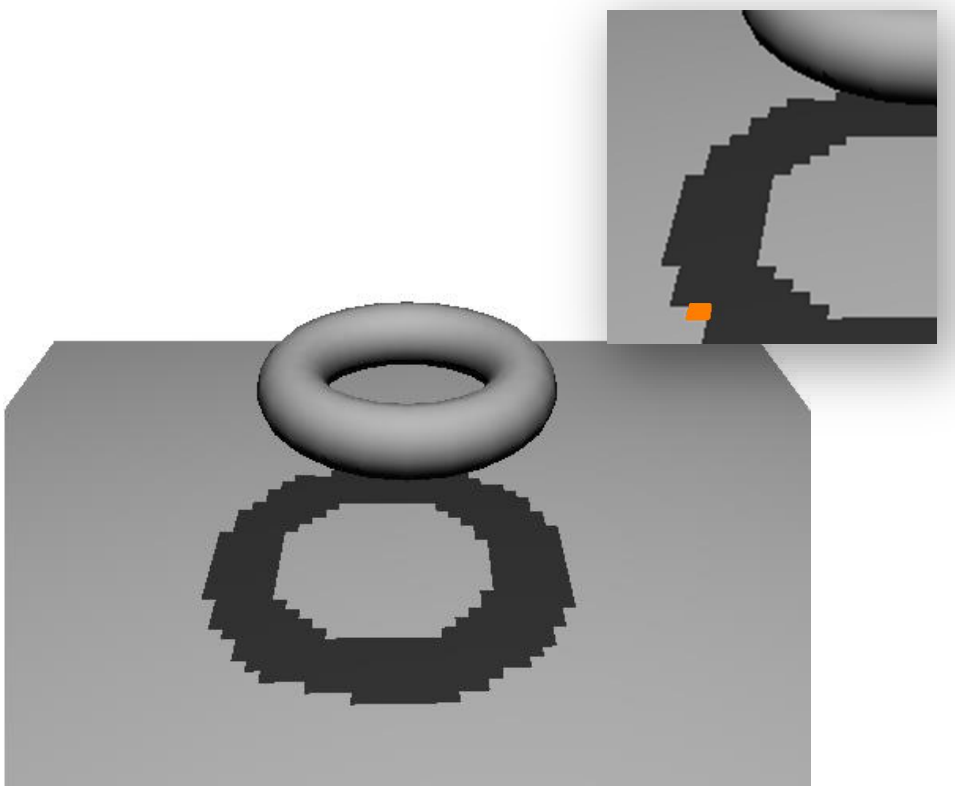


Shadow Maps

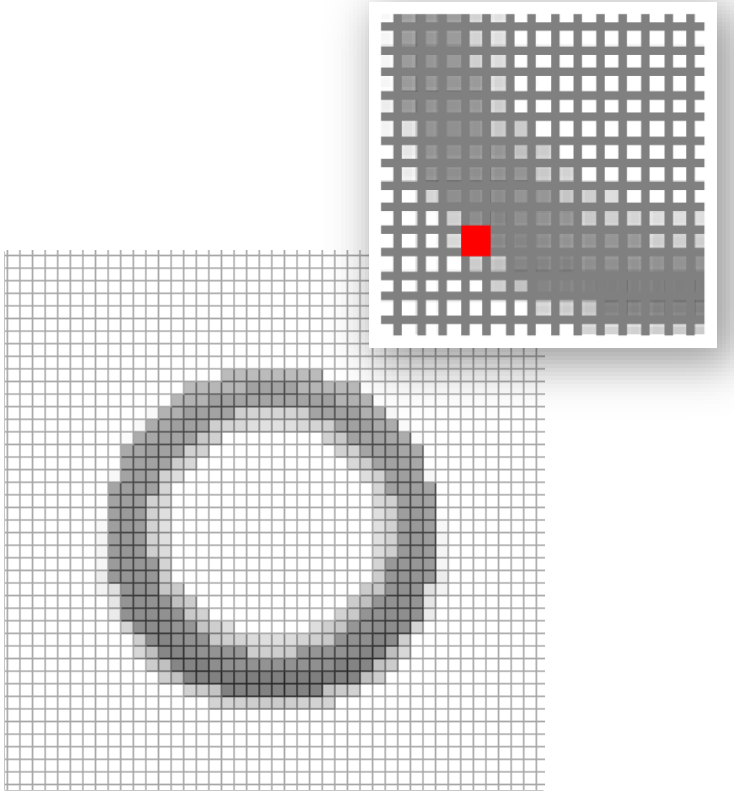
- Pros
  - Very efficient: “This is as fast as it gets”
- Cons…

Shadow Maps - Problems

- Low Shadow Map resolution results in jagged shadows



from viewpoint



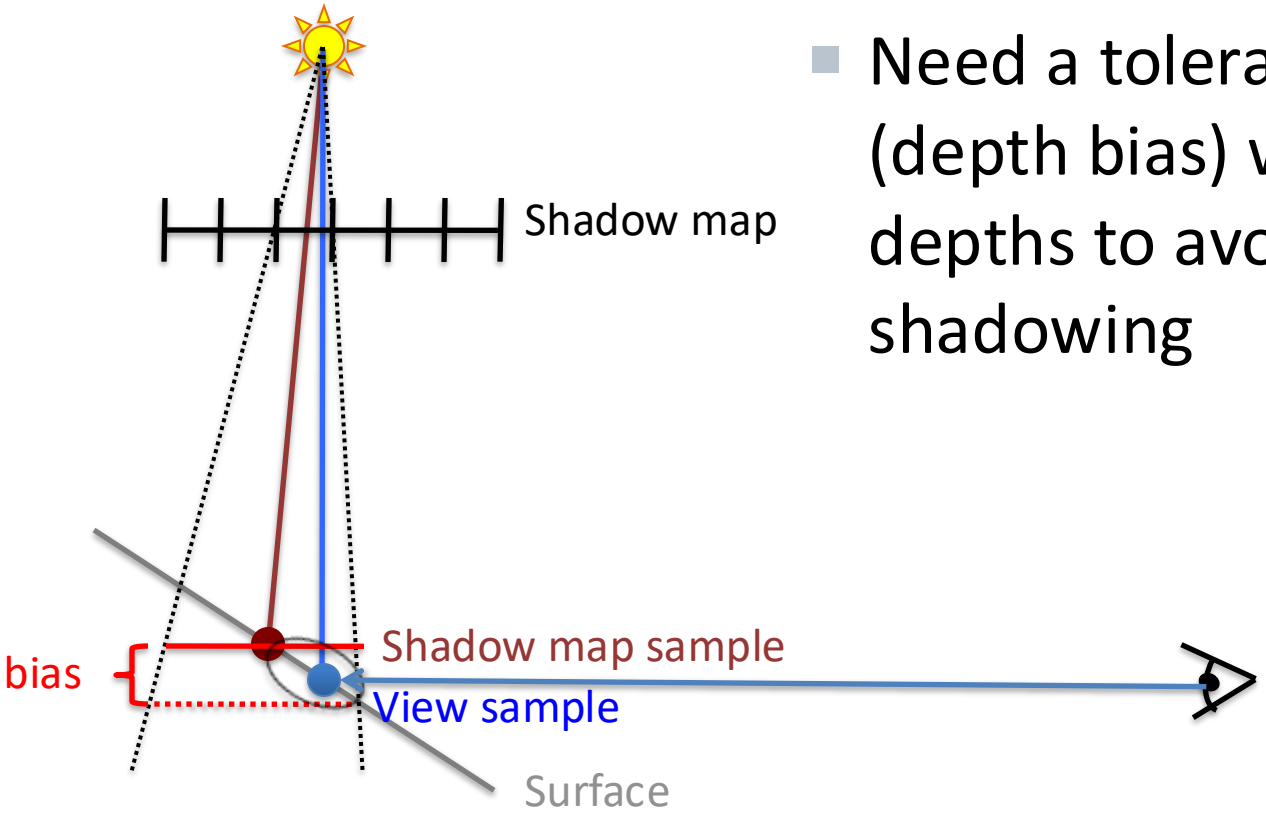
from light

Shadow Maps - Problems

In addition:

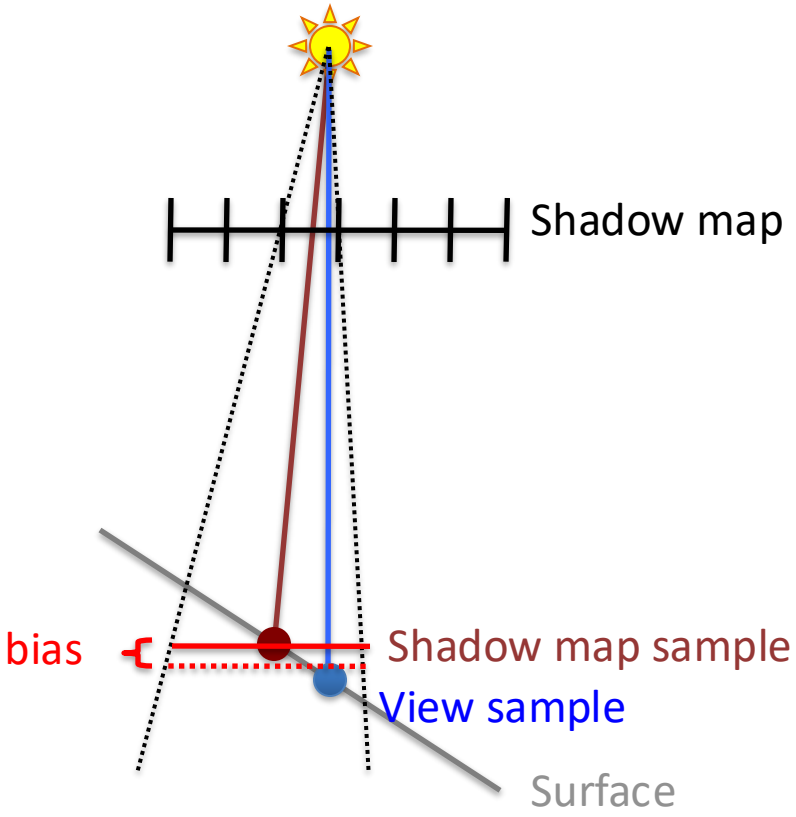
- A tolerance threshold (bias) needs to be tuned for each scene for the depth comparison

Bias

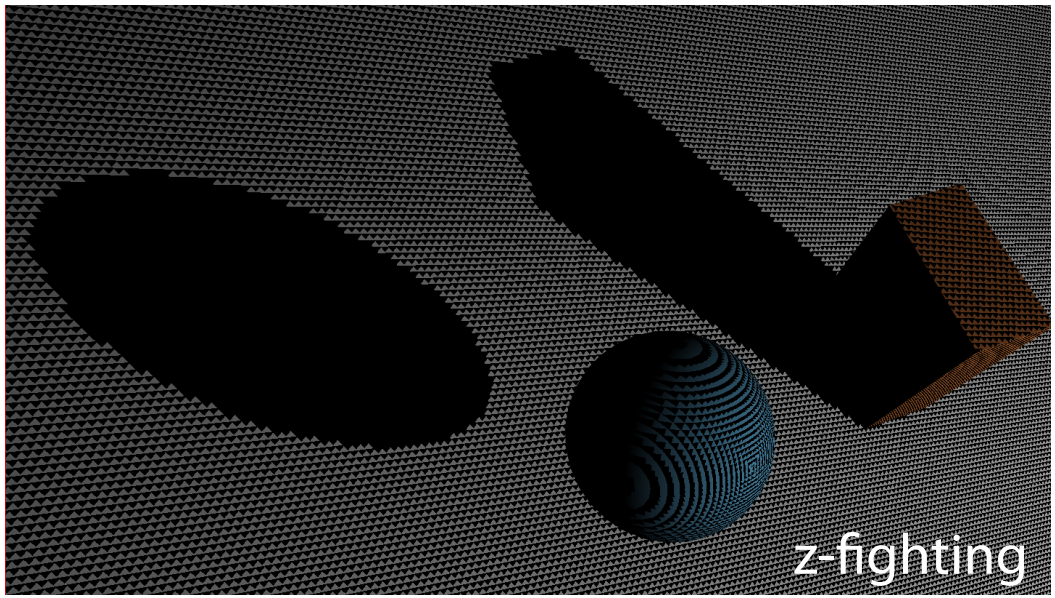


- Need a tolerance threshold (depth bias) when comparing depths to avoid surface self shadowing

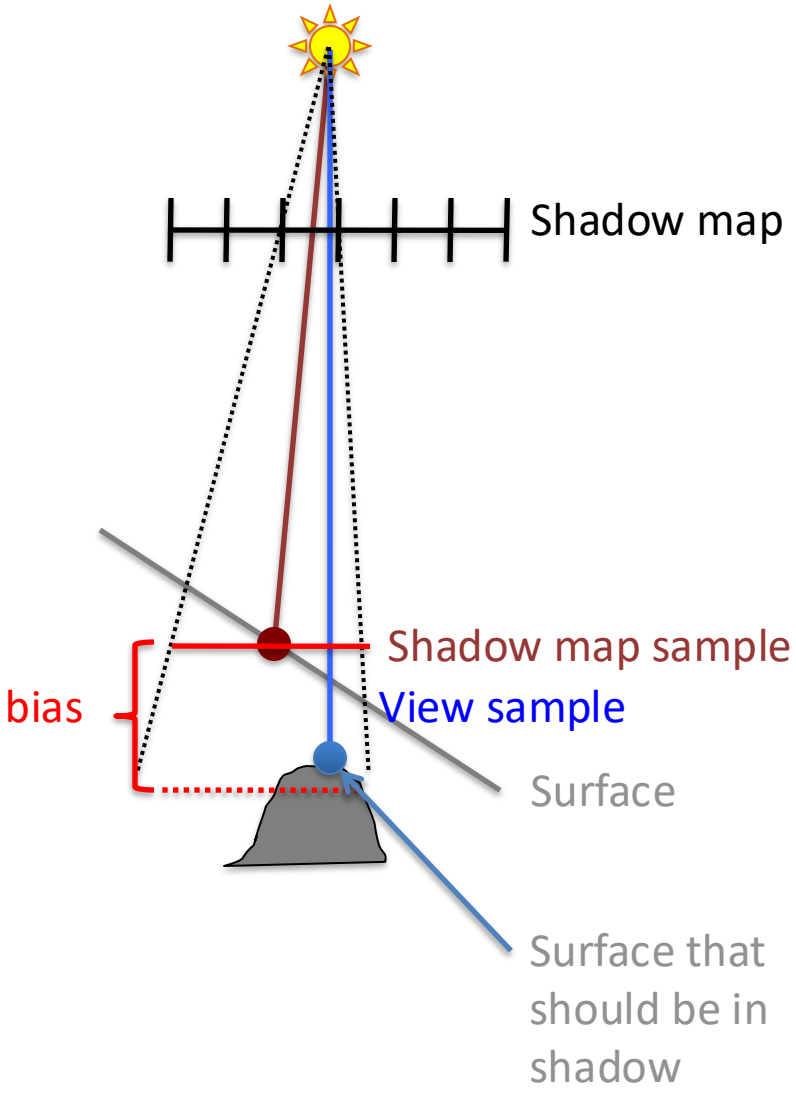
Bias



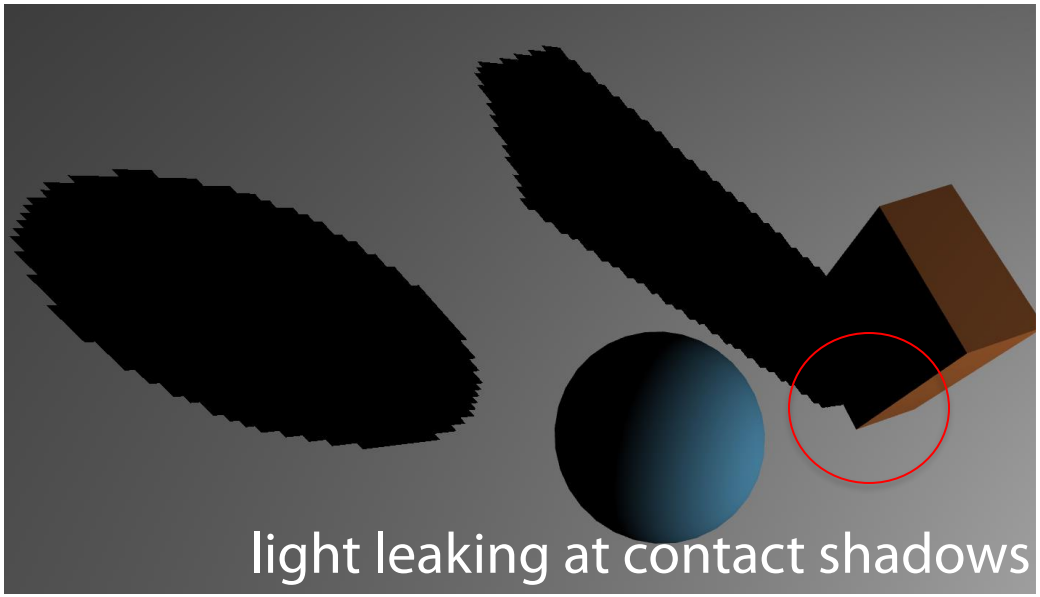
- Need a tolerance threshold (depth bias) when comparing depths to avoid surface self shadowing



Bias



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



Implementing Shadow Maps

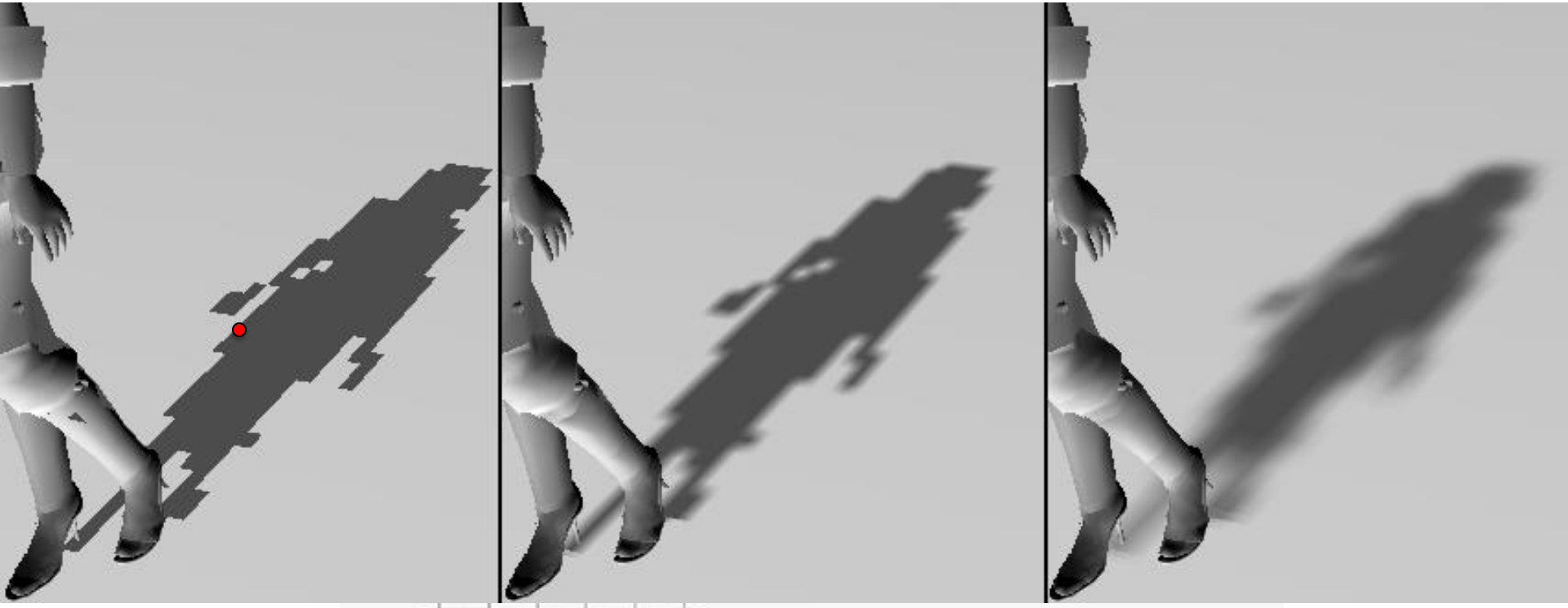
- See tutorial 6 on how to implement shadow maps in practice, as since 2012 ☺.
  - Changes every now and then, but algorithm stayed the same since 1978.

Percentage Closer Filtering

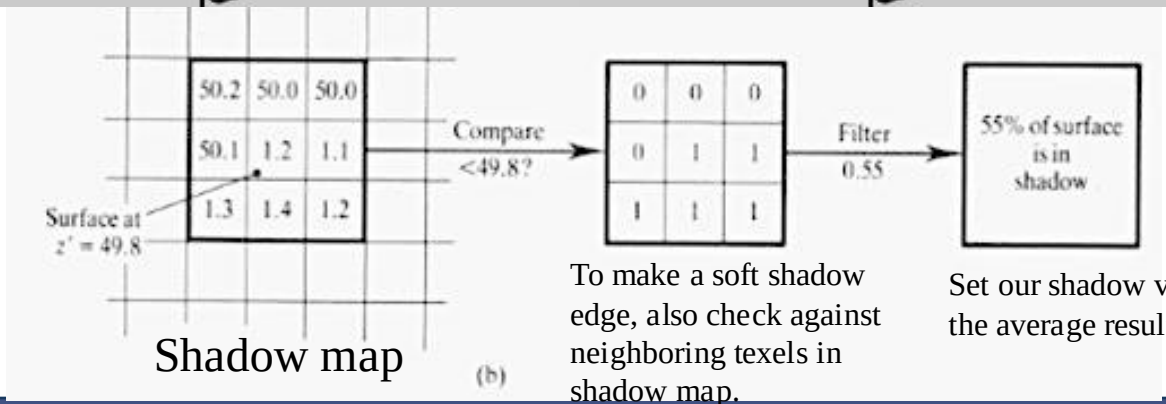
Percentage Closer Filtering

Percentage Closer Filtering

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.



Our fragment has z = 49.8 in light's space. That is further than shadow map's z-value for that location, so our fragment is in shadow.



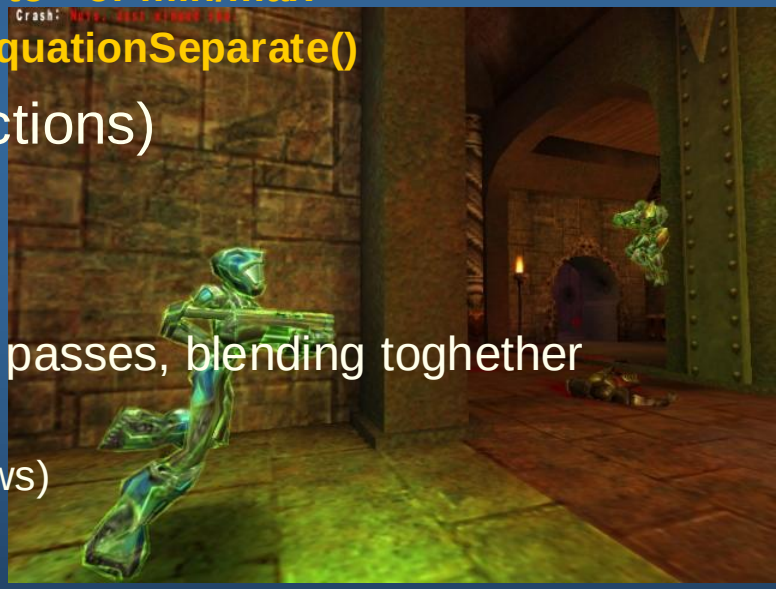
To make a soft shadow edge, also check against neighboring texels in shadow map.

Set our shadow value to the average 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 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)**

$$\mathbf{c}_o = \alpha \mathbf{c}_s + (1-\alpha)\mathbf{c}_d$$



Example of blending for Motion Blur

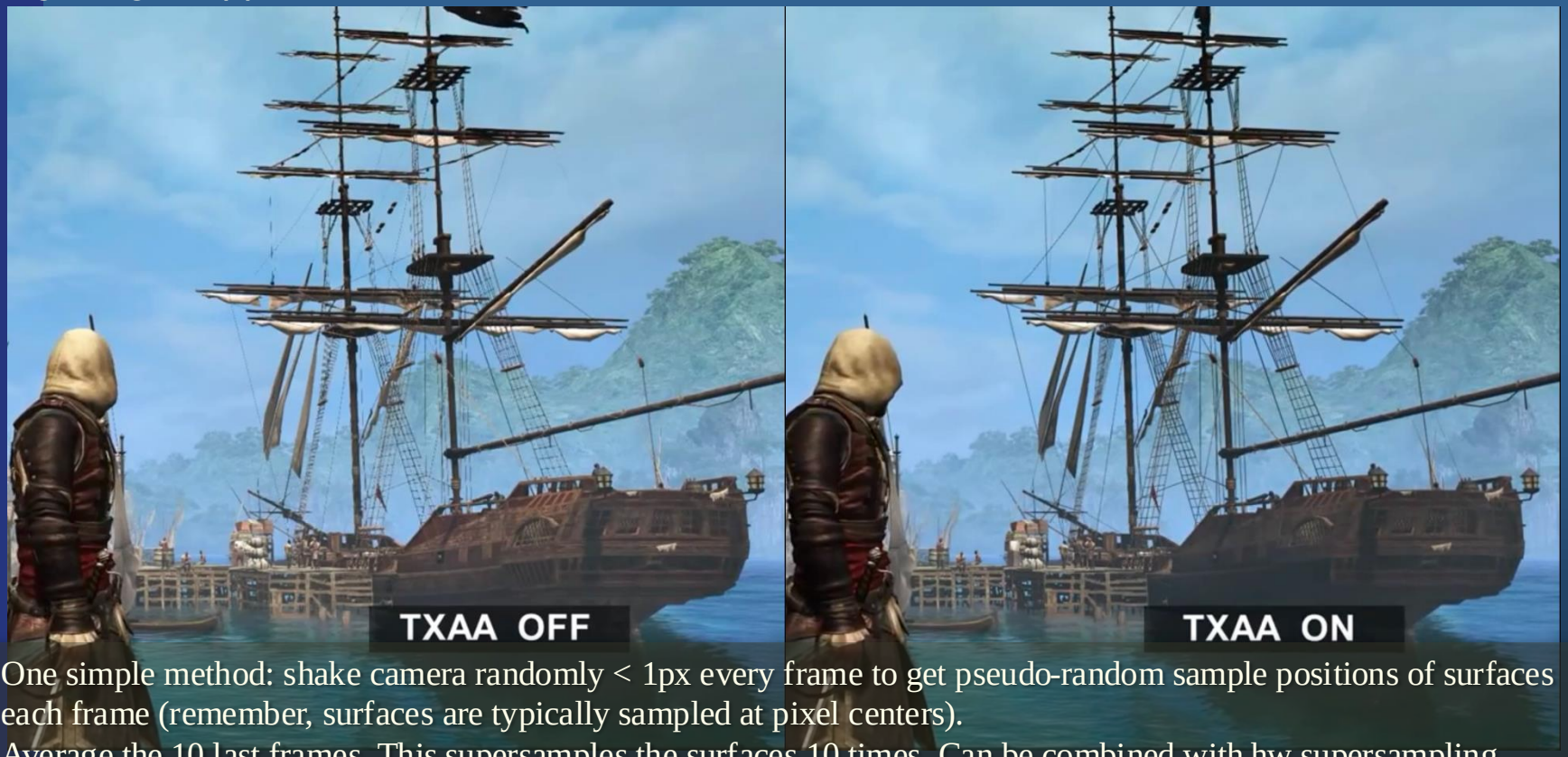
Possible with usage of e.g blending to 32-bit floating point rgb buffer and averaging result before displaying

Image courtesy Brostow and Essa

Temporal Anti-aliasing (TXAA)

- Assasin's Creed IV Black Flag

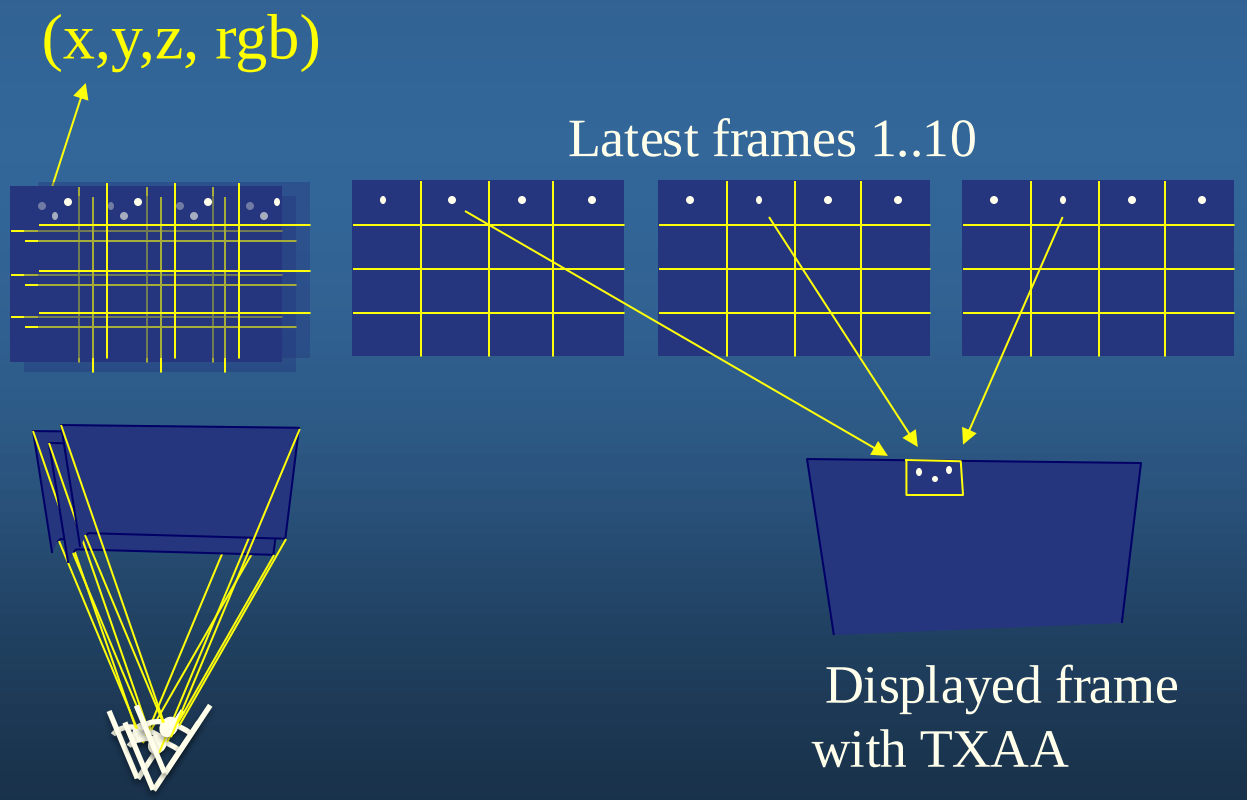
Temporal Anti-**aliasing** seeks to reduce or remove the effects of **temporal aliasing**. (A common example of **temporal aliasing** in film is the appearance of vehicle wheels travelling backwards, the so-called wagon-wheel effect.) **Temporal** anti-**aliasing** can also help to reduce jaggies, making 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 ∈[0,9]

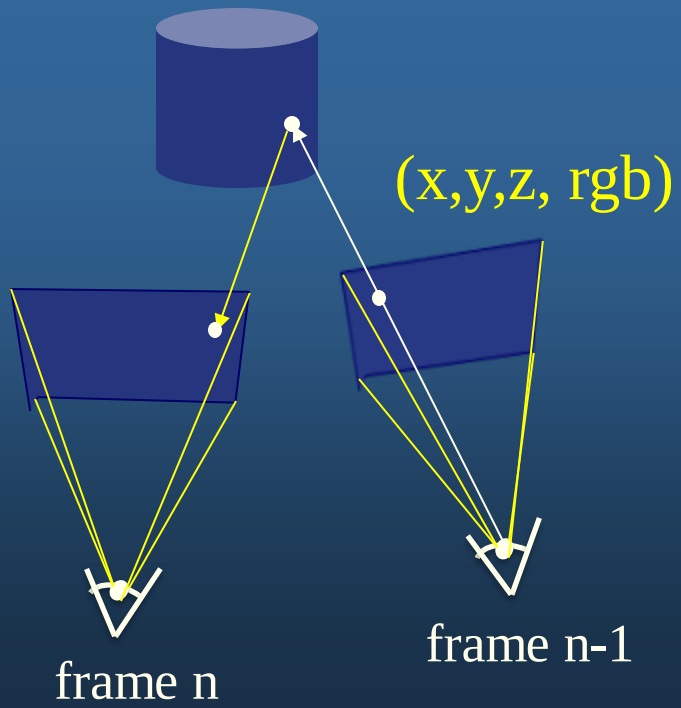
Temporal Anti-aliasing (TXAA)

- 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.



Temporal Anti-aliasing (TXAA)

- For moving cameras, project previous frames' pixel content to world-space and then into current frame's viewspace.
- For dynamic objects, also use their motion vectors to predict where their rasterized content from previous frames are in current frame.



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

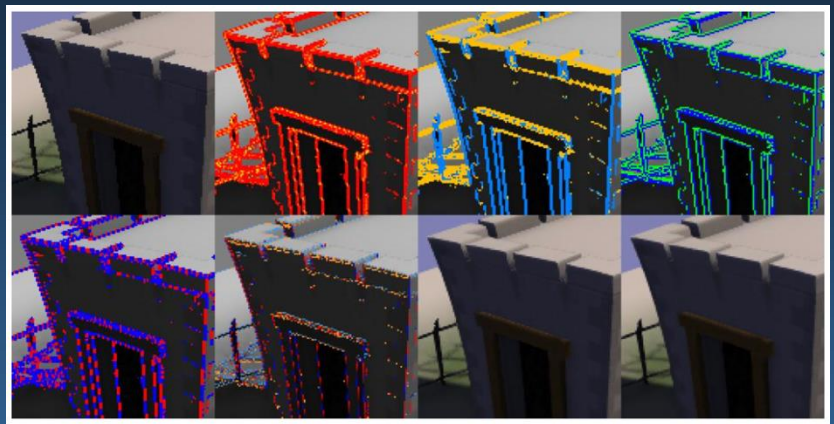


Figure 1: FXAA algorithm from right to left, top to bottom.

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)

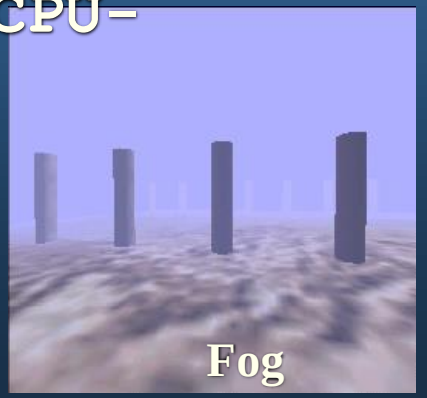
- **glFrameBufferTexture1/2/3D()**

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 stecilbuffer = 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: `gl_ClipDistance[]`
    - which are sent as outputs from the vertex shader to the fragment shader.
  - `glEnable(GL_CLIP_DISTANCEi)`
  - ***Gl_CullDistance*** – discards the whole triangle if fully outside
- 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

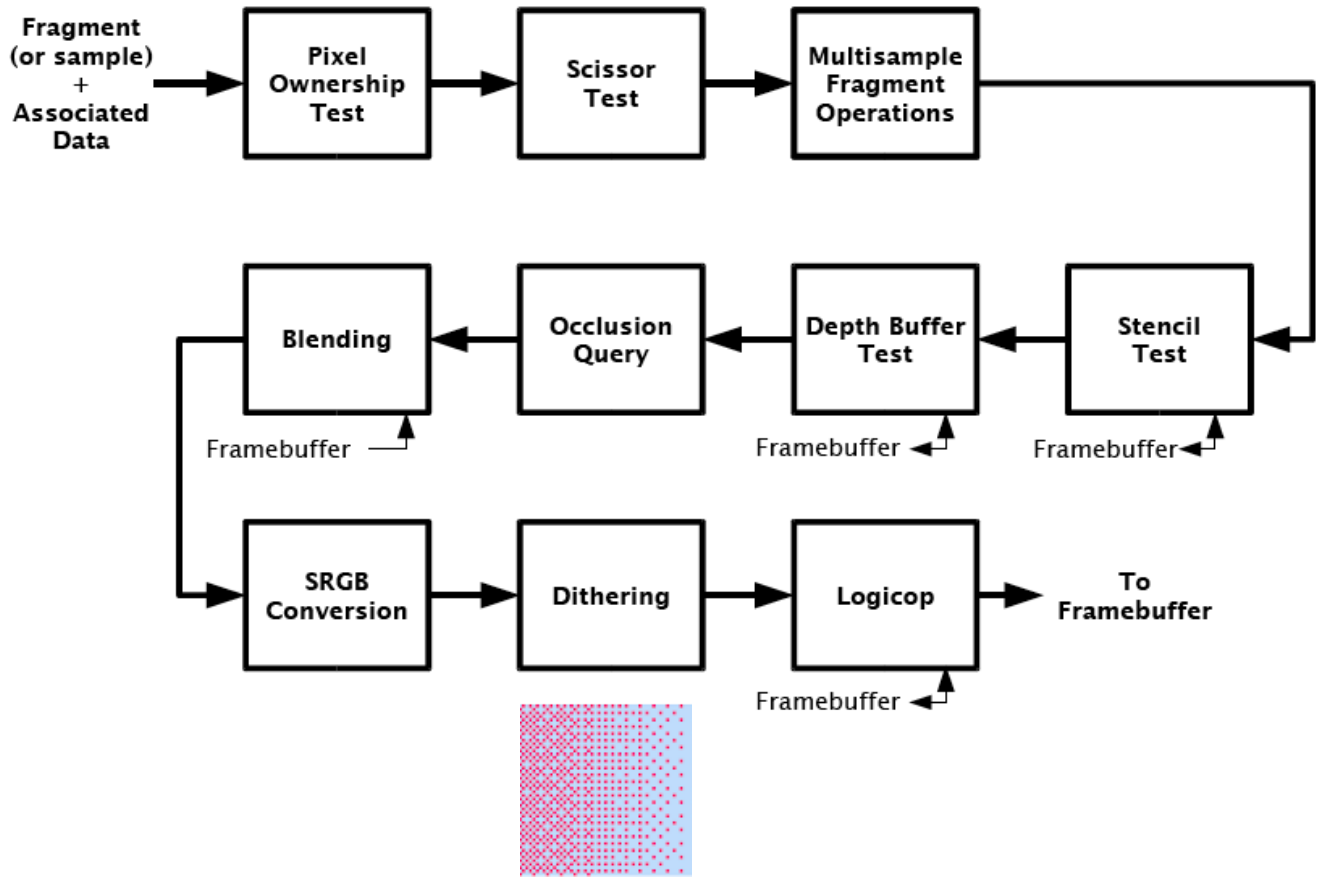


Figure 4.1. Per-fragment operations.

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-, tesselation-, 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:
  - **gTexParameterIivEXT = 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?