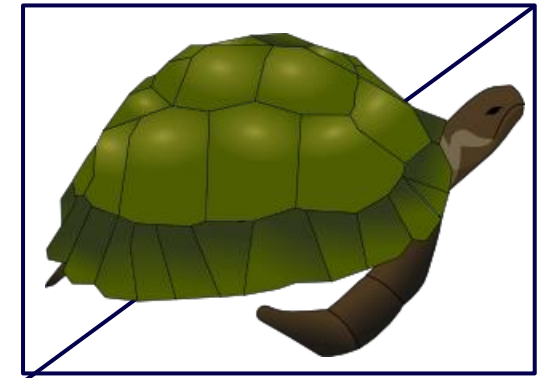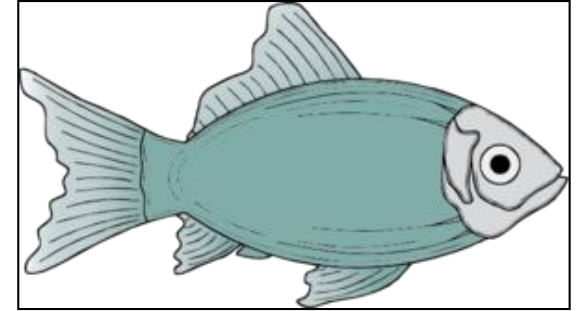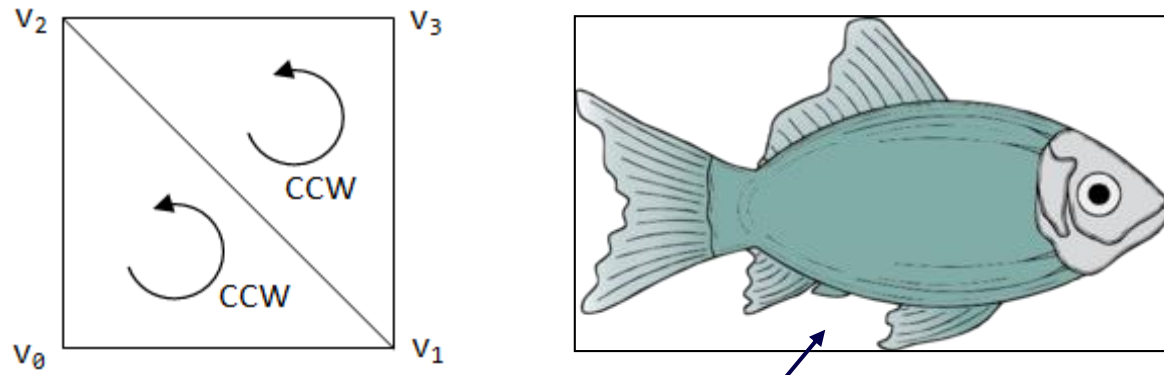# Advanced Rendering

Advanced OpenGL

SPRITES: Faking 2D Geometry

- Creating geometry is hard
- Creating texture is "easy"
- In 2D it is hard to see the difference

- SPRITE:
  - *Use basic geometry (rectangle = 2 triangles)*
  - *Texture the geometry (transparent background)*
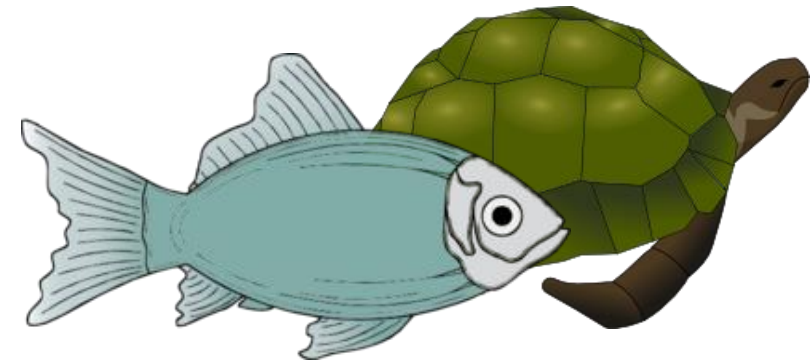 - *Use blending (more later) for color effects*

Sprite basics

A textured quad looks like fine-grained 2D geometry

Transparent with alpha = 0
e.g., color_RGBA = {1,1,1,0}

Proper occlusion despite
simple geometry

SPRITES: Creation

OpenGL initialization (once):
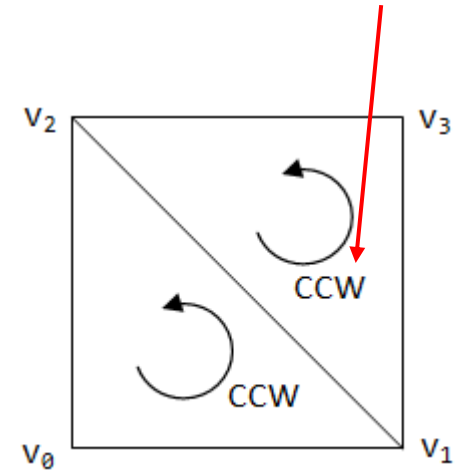
Create Quad Vertex Buffer

Counter-clockwise winding (CCW)

```
vec3 vertices[] = { v0, v1, v2, v3 };

glGenBuffers(1, &vbo);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBufferData(GL_ARRAY_BUFFER, vertices_size, vertices,
GL_STATIC_DRAW);
```

SPRITES: Creation

OpenGL initialization (once):

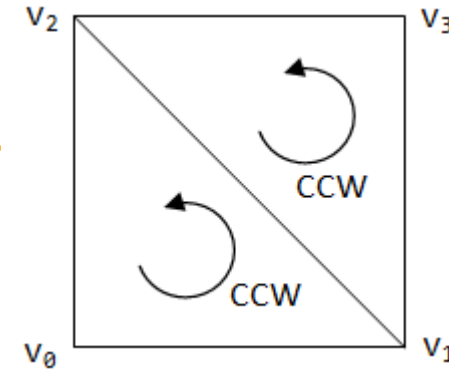### Create Quad Index Buffer

```
uint16_t indices[] = { 0, 1, 2, 1, 3, 2 };
Gluint ibo;
glGenBuffers(1, &ibo);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,ibo);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices_size, indices,
GL_STATIC_DRAW);
```

Load Texture

```
Gluint tex_id;
glGenTextures(1, &tex_id);
glBindTexture(GL_TEXTURE_2D, tex_id);
glTexImage2D(GL_TEXTURE_2D, GL_RGBA, width, height, .., tex_data);
```
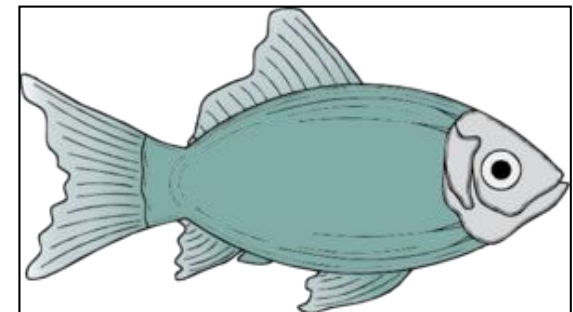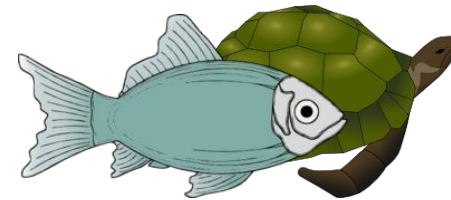
SPRITES: Rendering

OpenGL rendering (every frame):

Bind Buffers

```
glBindVertexArray(vao);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, ibo);
```

Enable Alpha Blending

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
// Alpha Channel Interpolation
// RGB_o = RGB_src * ALPHA_src + RGB_dst * (1 – ALPHA_src)
```

SPRITES: Rendering

Bind Texture

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texmesh.texture.texture_id);
```

Draw

```
glDrawElements(GL_TRIANGLES, 6, ..); // 6 is the number of indices
```

Sprite in Unreal Engine with paper2D:
https://www.youtube.com/watch?v=3GtGvNoL5UI&list=PLZlv_N0_O1gauJh
7mSr_67jqK42twB

1) Spr

Sprintes from LearnOpengl.com / show code

Color and Texture Mapping

- *How to map from a 2D texture to a 3D object that is projected onto a 2D scene?*

Scan Conversion/Rasterization

- Convert continuous 2D geometry to discrete

- Raster display – discrete grid of elements

- Terminology

  - **Screen Space:** *Discrete 2D Cartesian coordinate system of the screen pixels*
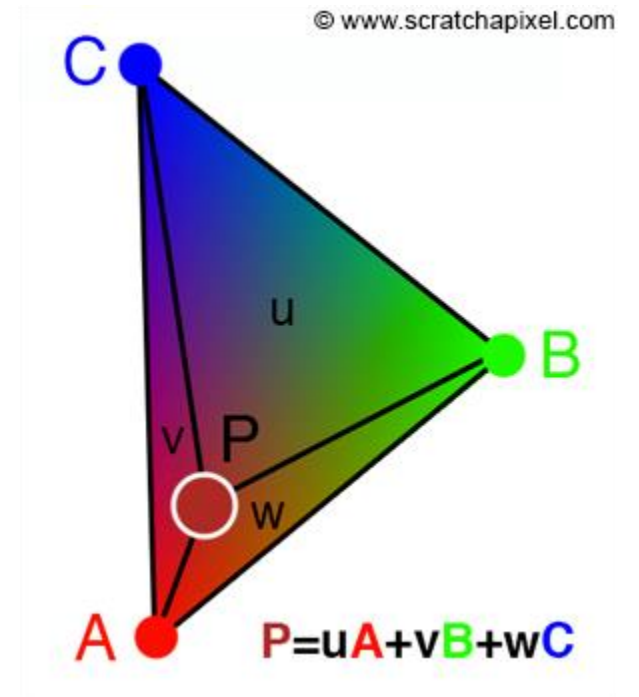
Scan Conversion
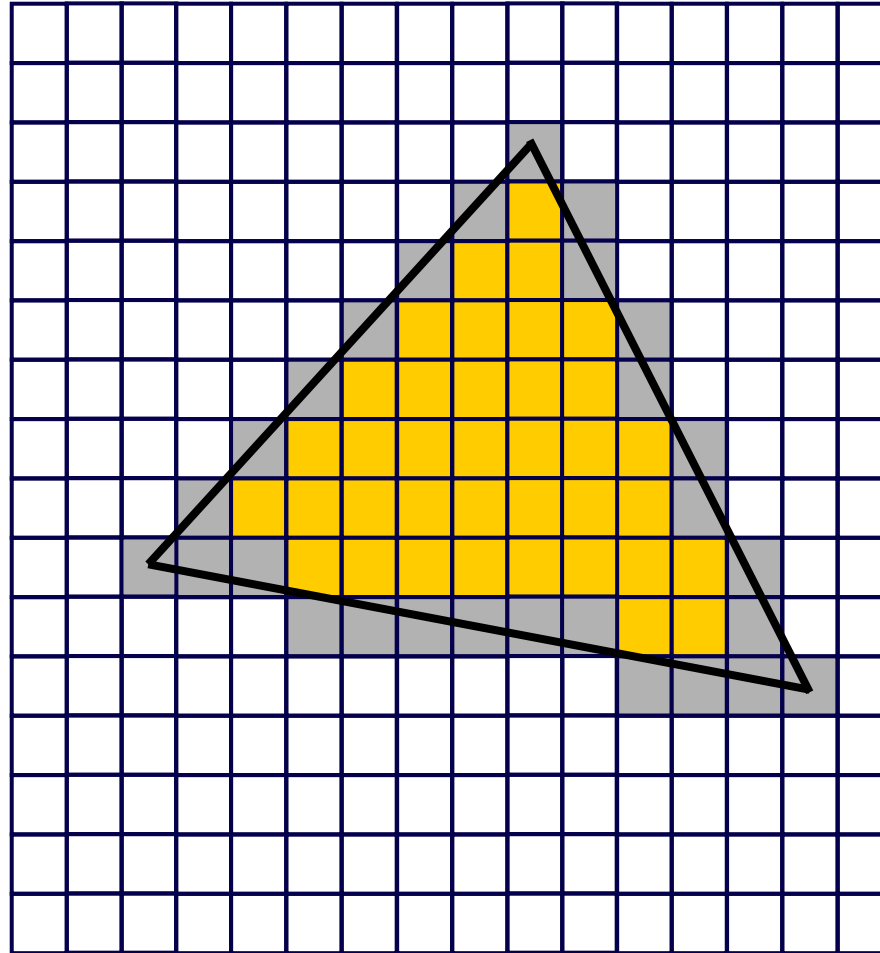
Self study:
Interpolation with barycentric coordinates

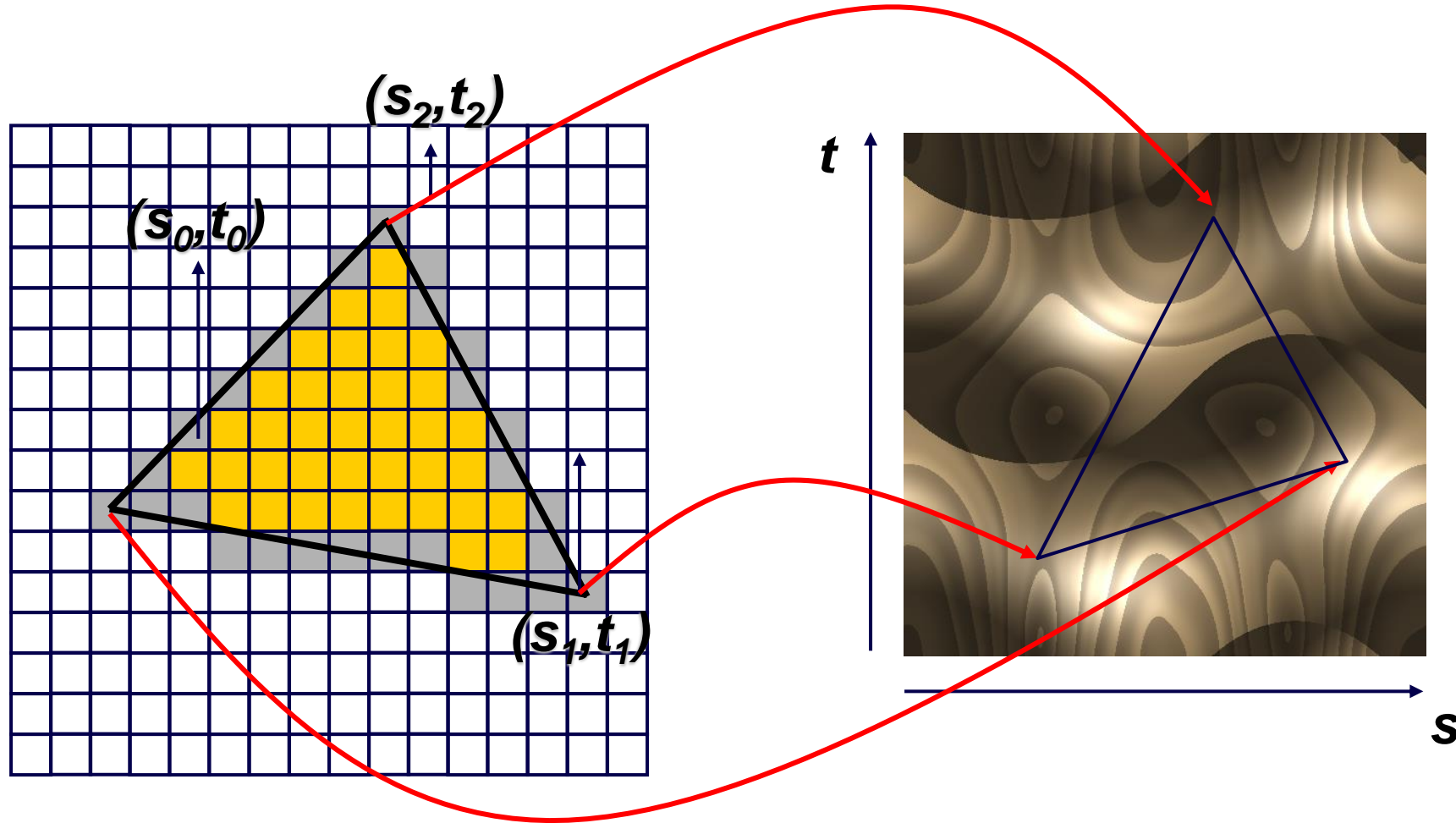- *linear combination of vertex properties*

  - *e.g., color, texture coordinate, surface normal/direction, …*

- *weights are proportional to the areas spanned by the sides to query point P*

© www.scratchapixel.com

$P = uA + vB + wC$
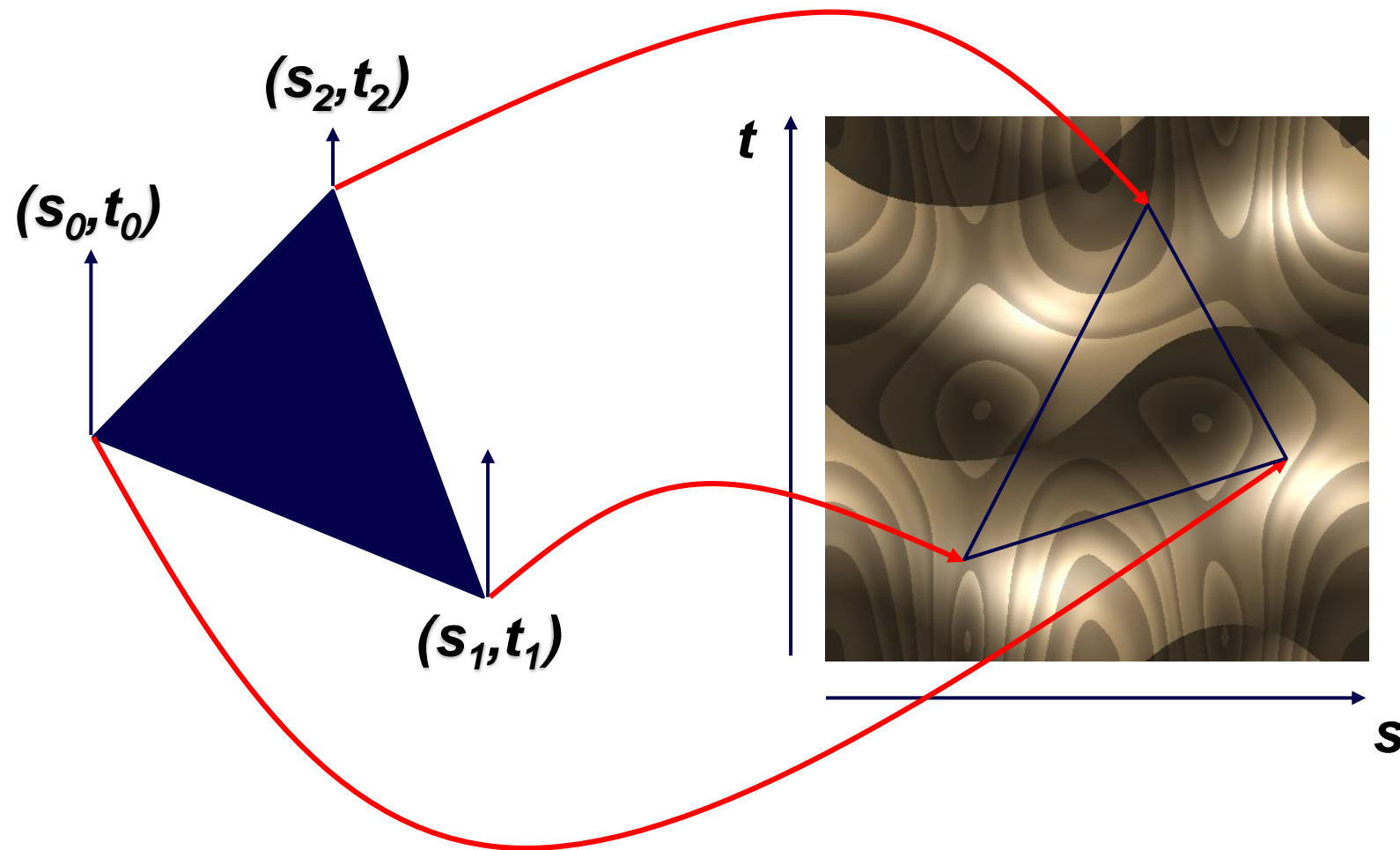
Texture mapping

Texture mapping

(s_2, t_2)

(s_0, t_0)

(s_1, t_1)

t

s

Texture mapping

(s_2,t_2)

(s_0,t_0)

(s_1,t_1)

t

s
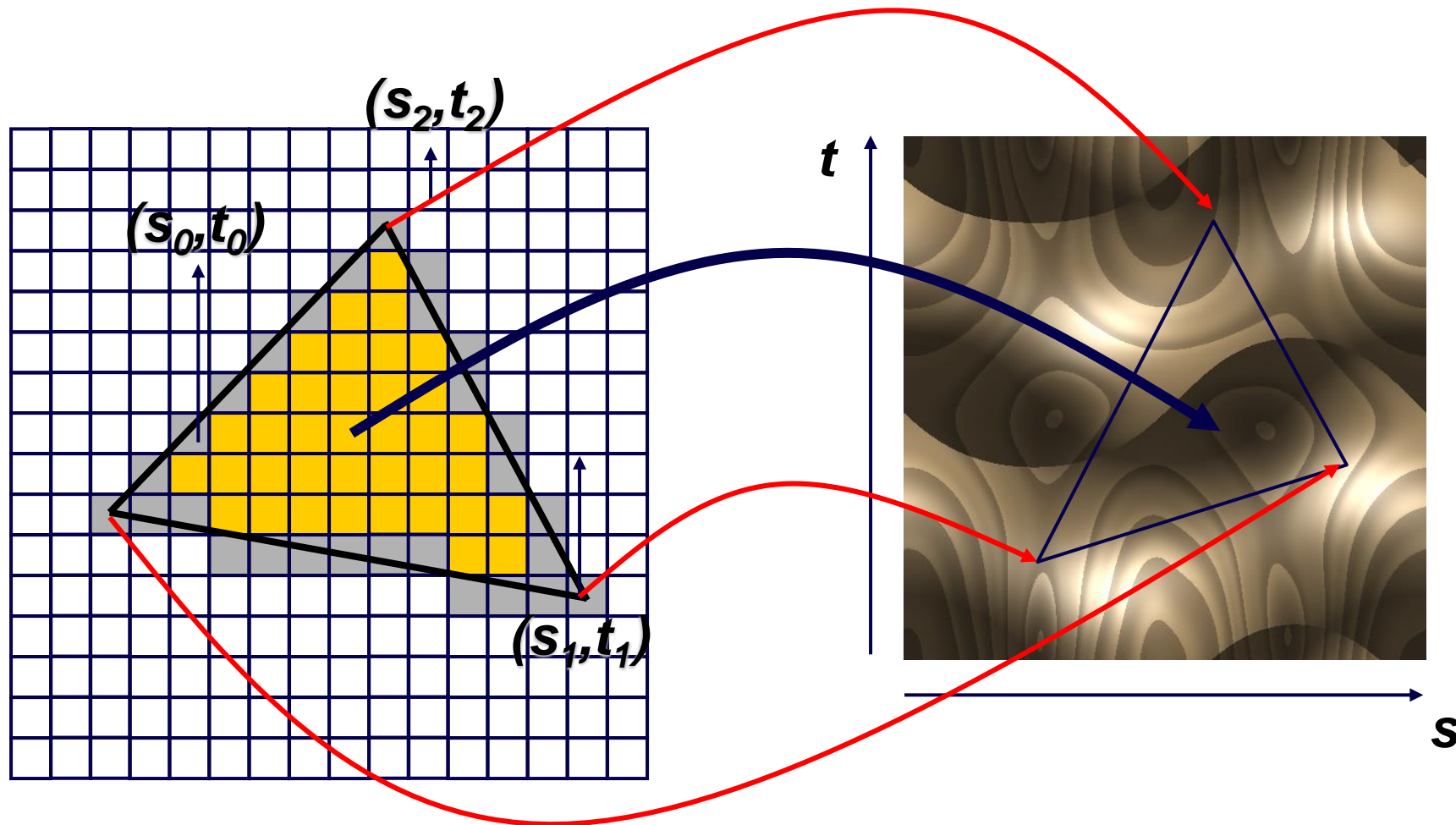
Blending

Blending:

- Fragments -> Pixels
- Draw from farthest to nearest
- No blending – replace previous color
- Blending: combine new & old values with some arithmetic operations
    - *Achieve transparency effects*

Frame Buffer : video memory on graphics board that holds resulting image & used to display it

Depth Test / Hidden Surface Removal

Remove occluded geometry

- Parts that are hidden behind other geometry

- For 2D (view parallel) shapes – use depth order

  - *draw objects back to front*

    - sort objects: furthest object first, closest object last

Self study: Alternative to ordering
Depth buffer with transparent sprites

- ***Fragment shader writes depth to the depth buffer***
  - *discard fragment if depth larger than current depth buffer (occluded)*
  - *alleviates the ordering of objects*

- **Issue, depth buffer written for fragments with alpha = 0**

- **Solution:**
  explicitly discard fragments
  with alpha < 0.5
  - *note, texture sample interpolation leads to non-binary values even if texture is either 0 or 1.*
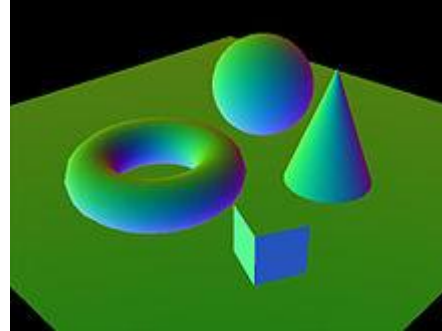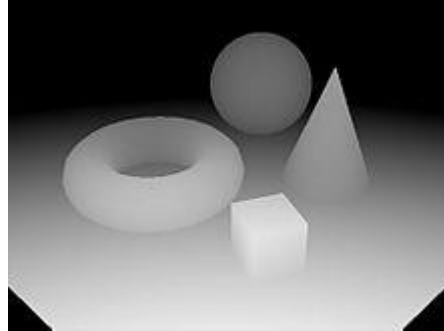
```
#version 330
in vec2 texCoord;
out vec4 outColor;
uniform sampler2D theTexture;

void main() {
  vec4 texel = texture(theTexture, texCoord);
  if(texel.a < 0.5)
    discard;
  outColor = texel;
}
```

Motivation

- *Deferred shading (a form of screen-space rendering)*

First rendering pass
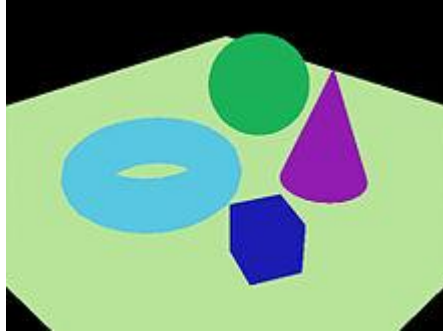
Second pass

Input →

- *or water effects*

First pass
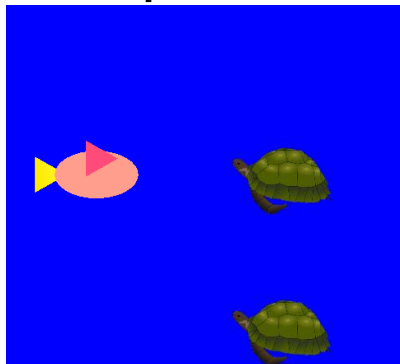
Second pass
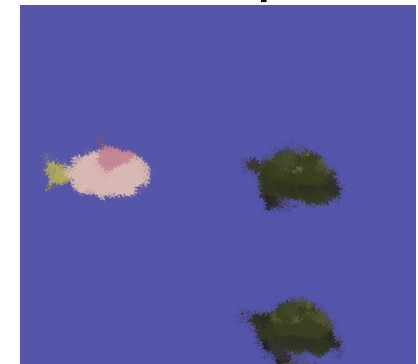
A few advanced examples

Blending
Sprite Sheets
Render to Texture
Post-processing Effects: Bloom

Blending

- **Controls how pixel color is blended into the FBO's Color Attachment**
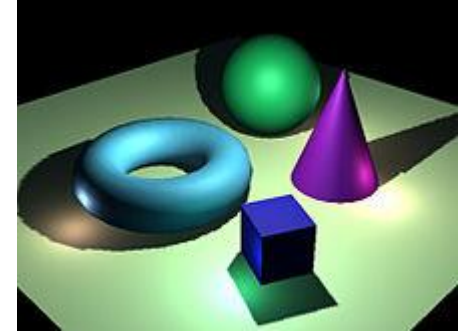
- **Control on factors and operation of the equation**

- **RGB and Alpha are controllabe separately**

(Source Color * Source Alpha) + (Dest Color * One)

$$RGB_o = RGB_{src} * F_{src} \; [+ \; - \; / \; *] \; RGB_{dst} * F_{dst}$$
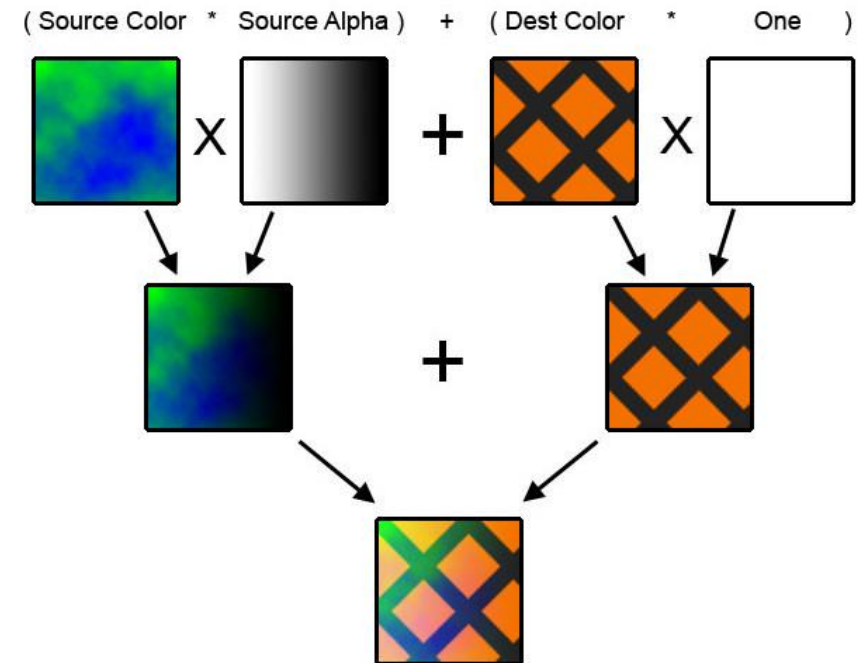
Cloud (source) on top of grid (dest)

Blending: Example Presets

- **Additive Blending**

```
//---
// RGB_o = RGB_src + RGB_dst
glEnable(GL_BLEND);
glBlendFunc(GL_ONE, GL_ONE);
```

- **Alpha Blending**

```
//---
// RGB_o = RGB_src * ALPHA_src + RGB_dst * (1 - ALPHA_src)
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Sprite Sheets

- **Compact (and fast) approach for 2D animations**

- **Every frame only a region of the original Texture is rendered**

- **Texture Coordinates are updated as clock ticks**

- **Does not require dynamic VBOs**

Time
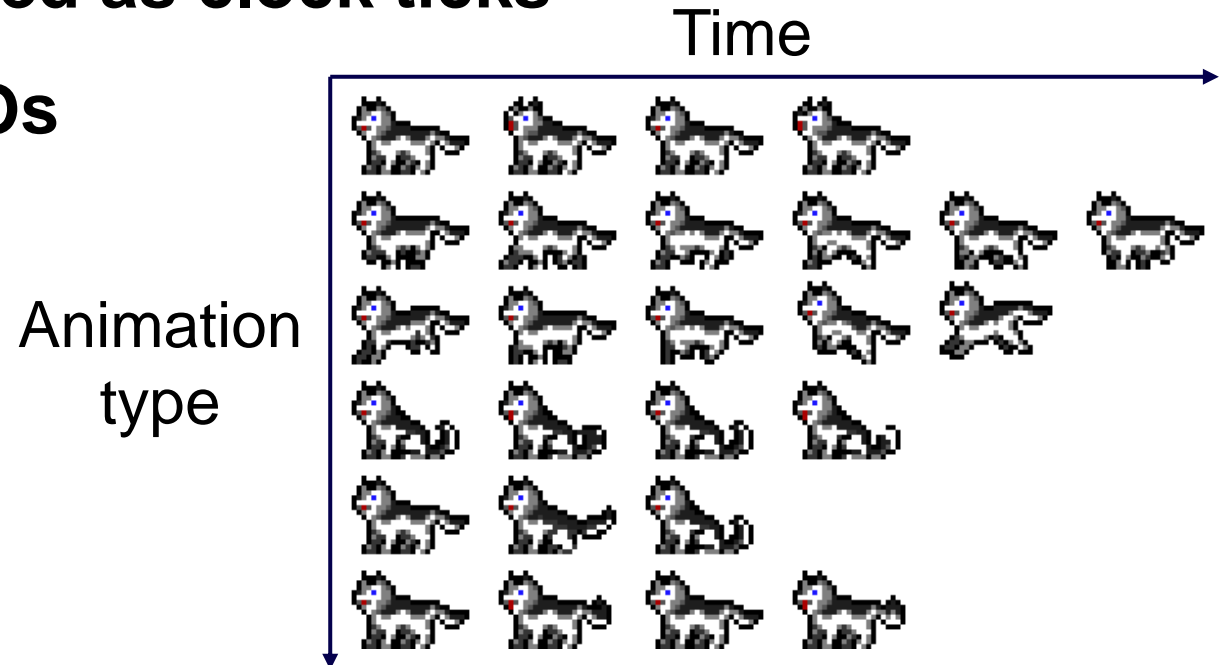
Animation
type

Sprite Sheets: Example

```
// APPLICATION
void load() {
    // ANIMATION_FRAME_[W|H] is in texture coordinates range [0, 1]
    vertices[0].texcoord = (0, 0);
    vertices[1].texcoord = (ANIMATION_FRAME_W, 0);
    vertices[2].texcoord = (ANIMATION_FRAME_W, ANIMATION_FRAME_H);
    vertices[3].texcoord = (0, ANIMATION_FRAME_H);
}

void update(float ms) {
    elapsed_time += ms;
    if (elapsed_time > ANIMATION_SPEED)
        frame = (frame+1)%NUM_ANIMATION_FRAMES;
}

void render() {
    glUniform1i(shader_program, &frame);
    ..
}

// SHADER
uniform vec2 texcoord_in; // Attribute coming from geometry (.texcoord)

void main() {
    texcoord = texcoord_in;

    // Sliding coordinates along X direction
    texcoord.x += ANIMATION_FRAME_W * frame;
}
```

Render To Texture

- **Building block of any multipass pipeline**
- **Just putting two concepts together..**

  - First Pass: Pixel colors are written to the FBO's Color Attachment
  - Second Pass: The same Texture can be bound and used by Samplers

```
// When Loading
render_target = create_texture(screen_resolution)  // Create texture (Usually with same screen resolution)
fbo = create_fbo(render_target) // Bind <render_target> as <fbo>'s color attachment

// First pass
bind_fbo(fbo)
draw_first_pass()

// Second pass
bind_fbo(0) // Reset to default FBO (Window)
bind_texture(render_target) // You can use <render_target> as you would you any other texture
draw_second_pass()
```
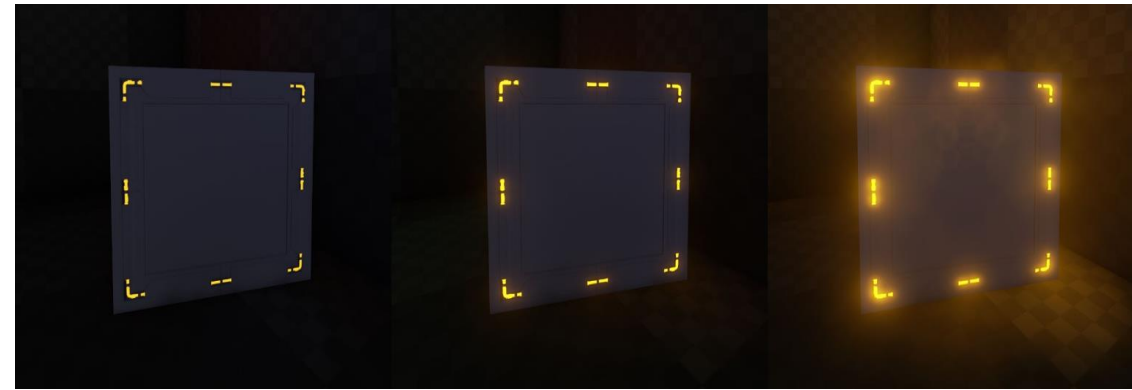
Post-processing: Bloom

- **Fullscreen Effect to highlight bright areas of the picture**

- **Post-processing: Operates on Images after the scene has been rendered**

- **High level overview:**

  1. Render scene to texture
  2. Extract bright regions by thresholding
  3. Gaussian blur pass on the bright regions
  4. Combine original texture and highlights texture with additive blending

Post-processing: Bloom

Threshold

Blur

Sum
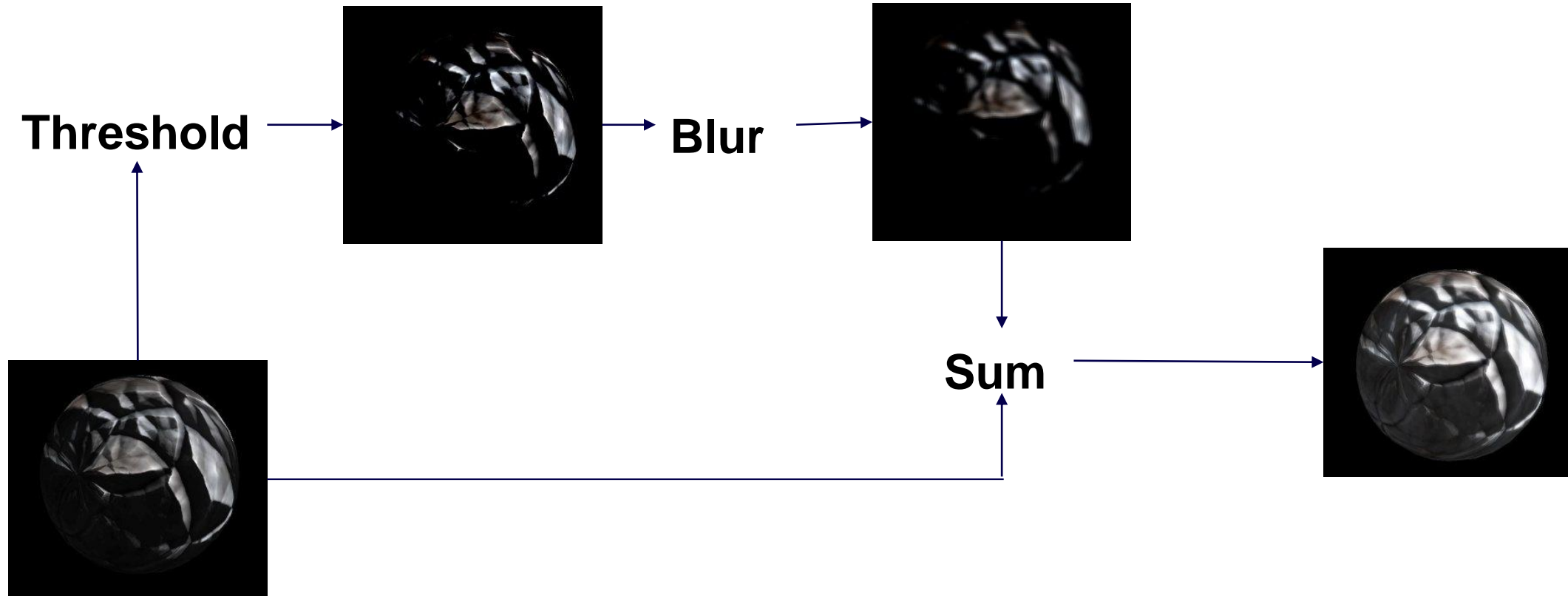
Post-processing: Bloom

```
GLuint original_rt, bright_rt, blur_rt;

bind_fbo(original_rt);
render_scene(original_rt);

bind_fbo(bright_rt);
threshold(original_rt); // Only keep pixels brighter than threshold

bind_fb(blur_rt);
gaussian_blur(bright_rt); // Blur bright regions

bind_fbo(0); // Writing to window's framebuffer
add(blur_rt, original_rt);
```

Self study: Post-processing: Bloom + DEMO from learnopengl.com

As many details have been skipped, here are a couple of hints:

- A fullscreen effect is achieved by rendering a textured quad with the same dimensions as the screen. No need for any camera or projection matrix as you already know that you want the vertices to correspond to the corners of the screen.

- Thresholding bright areas can be achieved in the fragment shader with something as simple as: `return Intensity > Threshold ? Color : 0.0;`
  Where `Intensity` is some function of the pixel's RGB values. You can start from max component, average, or explore other color space.

- Regarding Gaussian Blur (or Bloom altogether) there are lots of online resources of various quality.
  A suggested place to start for tutorials is **https://learnopengl.com**.
  The standard reference book for real-time rendering is "Real-Time Rendering" (**http://www.realtimerendering.com/**)

DEMO and talk: Deferred Rendering

https://www.youtube.com/watch?v=n5OiqJP2f7w

https://www.youtube.com/watch?v=0ckE-CZpXAo