centre adscrit a:

Computer Graphics

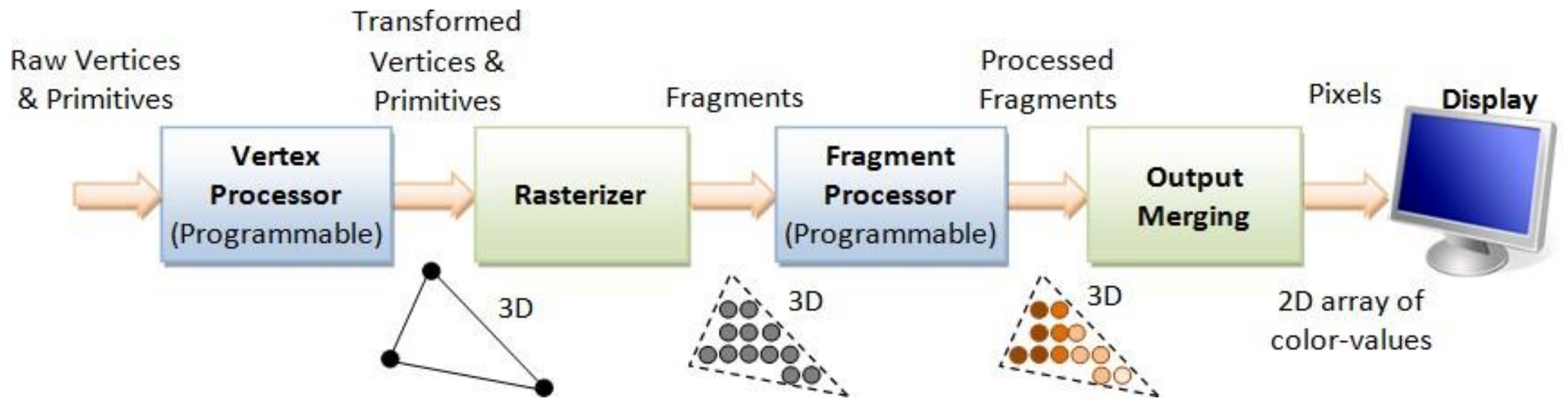
8.Non Realistic Rendering – Part 2

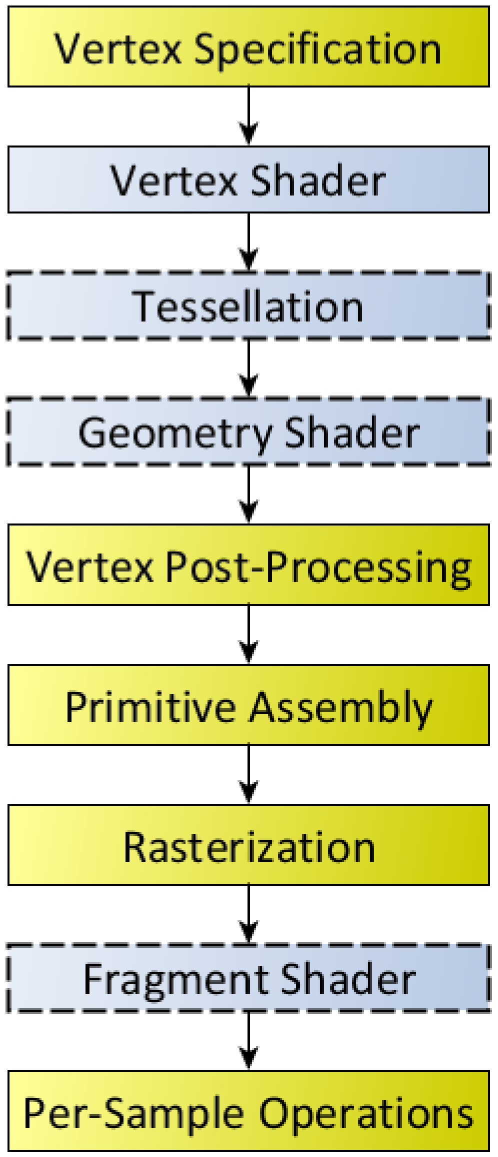
Dr Joan Llobera – joanllobera@enti.cat
Dr Jesus Ojeda – jesusojeda@enti.cat

Outline

1. Review of the graphics pipeline
2. Per sample operations
3. Depth, Scissor, Stencil Buffers
4. Contour Rendering

1. Reminder of the graphics pipeline


Raw Vertices & Primitives
Vertex Processor (Programmable)
Transformed Vertices & Primitives
Rasterizer
3D
Fragments
Fragment Processor (Programmable)
3D
Processed Fragments
Output Merging
3D
Pixels
Display
2D array of color-values



2. Per-sample operations

There are others, but here we shall only consider 3 of them

- Depth test
- Scissor test
- Stencil test

3. Intro to the depth buffer

Name

gl_FragCoord — contains the window-relative coordinates of the current fragment

Declaration

in vec4 gl_FragCoord ;

Description

Available only in the fragment language, gl_FragCoord is an input variable that contains the window relative coordinate (x, y, z, 1/w) values for the fragment. If multi-sampling, this value can be for any location within the pixel, or one of the fragment samples. This value is the result of fixed functionality that interpolates primitives after vertex processing to generate fragments. The z component is the depth value that would be used for the fragment's depth if no shader contained any writes to gl_FragDepth.

Name

gl_FragDepth — establishes a depth value for the current fragment

Declaration

out float gl_FragDepth ;

Description

Available only in the fragment language, gl_FragDepth is an output variable that is used to establish the depth value for the current fragment. If depth buffering is enabled and no shader writes to gl_FragDepth, then the fixed function value for depth will be used (this value is contained in the z component of gl_FragCoord) otherwise, the value written to gl_FragDepth is used.

3. Intro to the depth buffer

The depth test, when enabled, allows a fragment to be culled based on a conditional test between the fragment's depth value and the depth value stored in the current Depth Buffer at that fragment's sample position. This is useful to cause geometry to be hidden behind other geometry. The closer geometry lays down depth values that mask the rendering of any fragments behind them, using the proper depth test condition.

To enable depth testing:

```
glEnable(GL_DEPTH_TEST);
```

You should probably update the depth test at each rendering iteration. Therefore, you will need:

```
glClear(GL_COLOR_BUFFER_BIT
| GL_DEPTH_BUFFER_BIT);
```

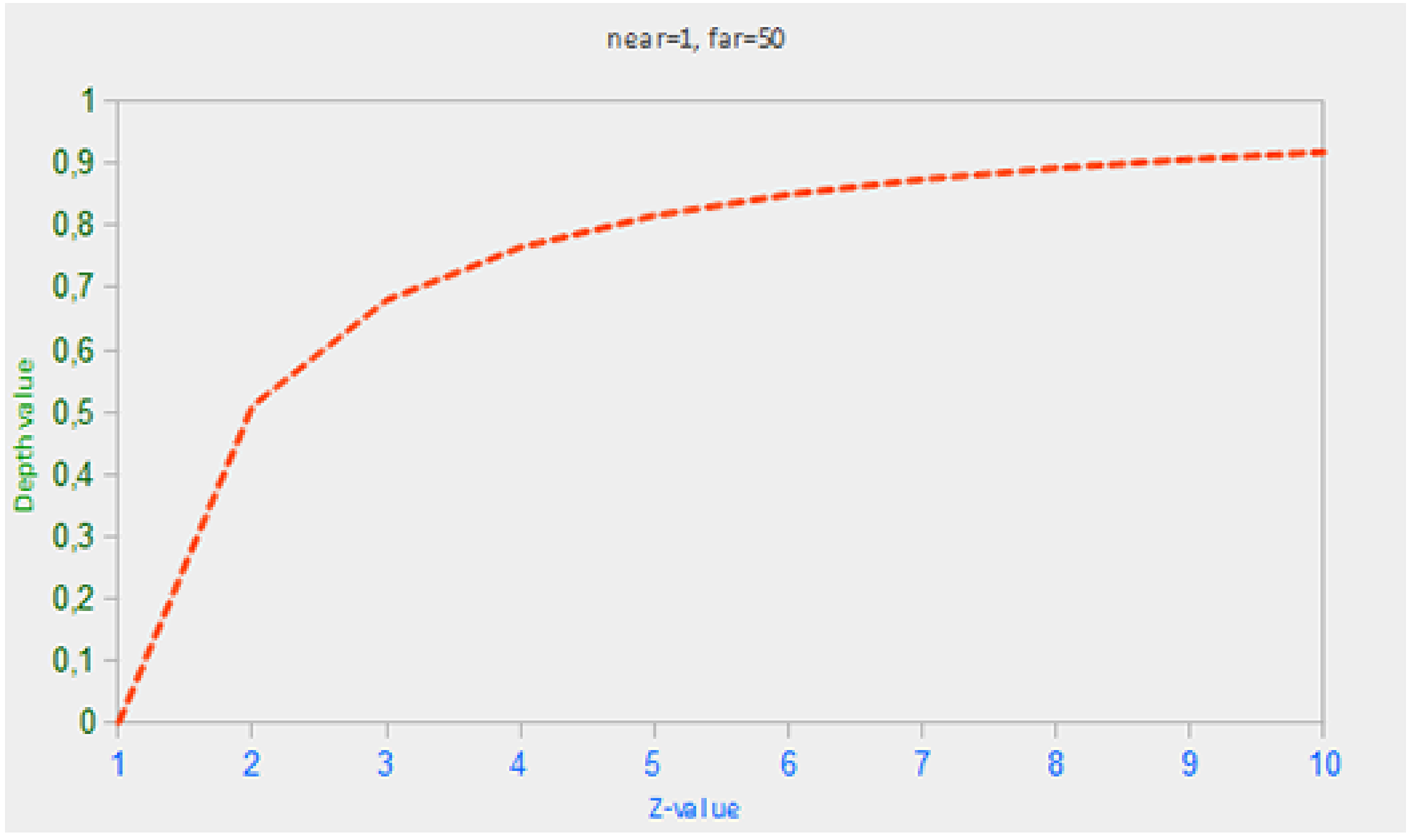
To render the depth buffer, in the fragment shader:

```
void main() { FragColor =
vec4(vec3(gl_FragCoord.z), 1.0);
}
```

3. Intro to the depth buffer

Beware!

The z buffer is not linear in precision: closer implies more precision



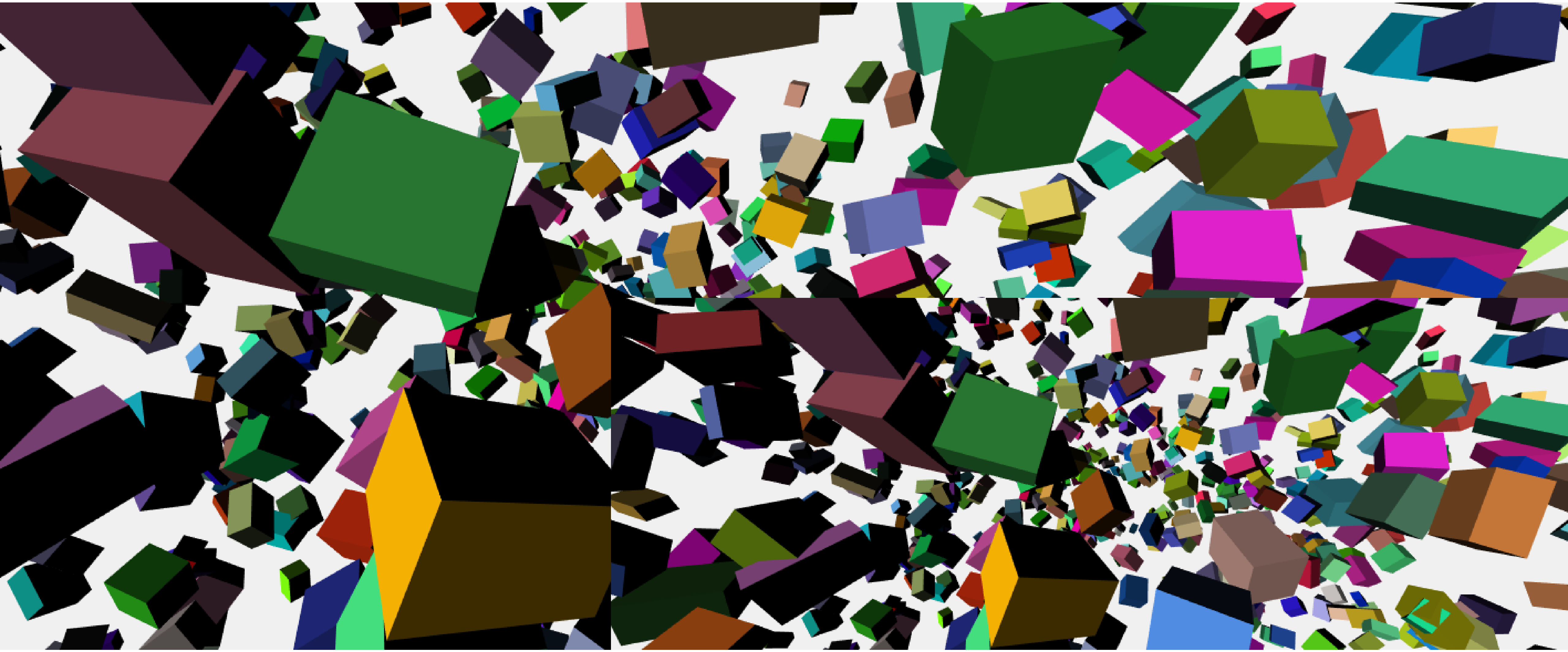
Source: https://learnopengl.com/Advanced-OpenGL/Depth-testing

3. Intro to Scissor tests

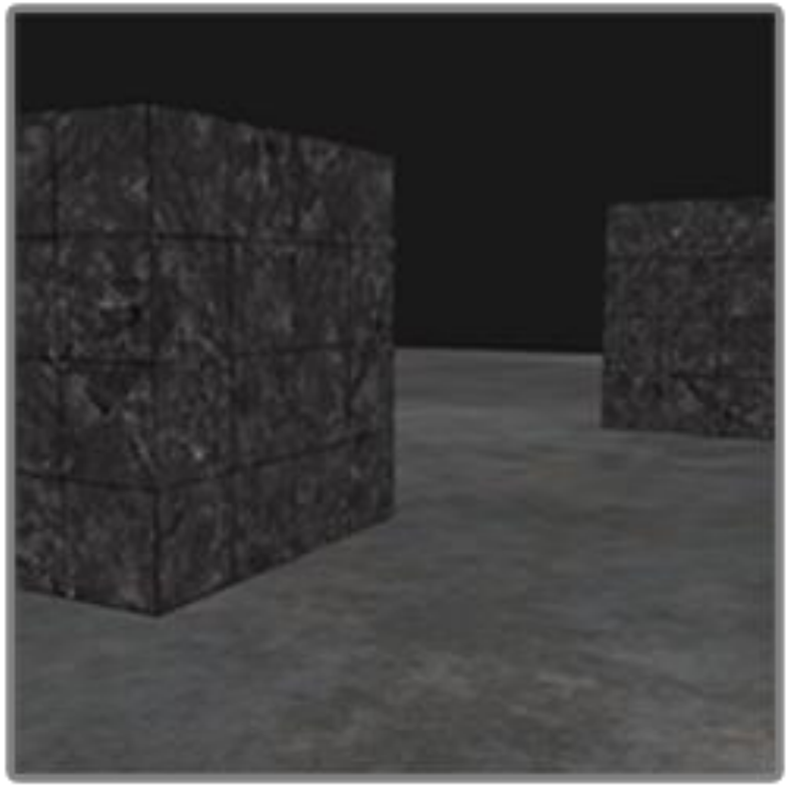
- It allows selecting a part of the rendering window

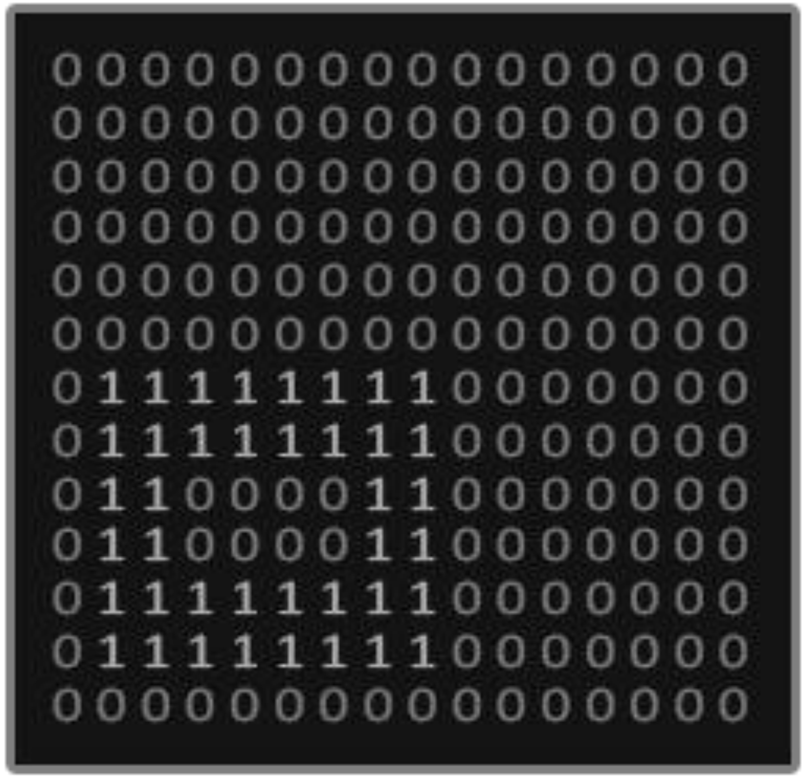
```
void glScissor(GLint x,
GLint y, GLsizei width,
GLsizei height);
```

- It is particularly useful when setting up multiple viewports → scissor arrays



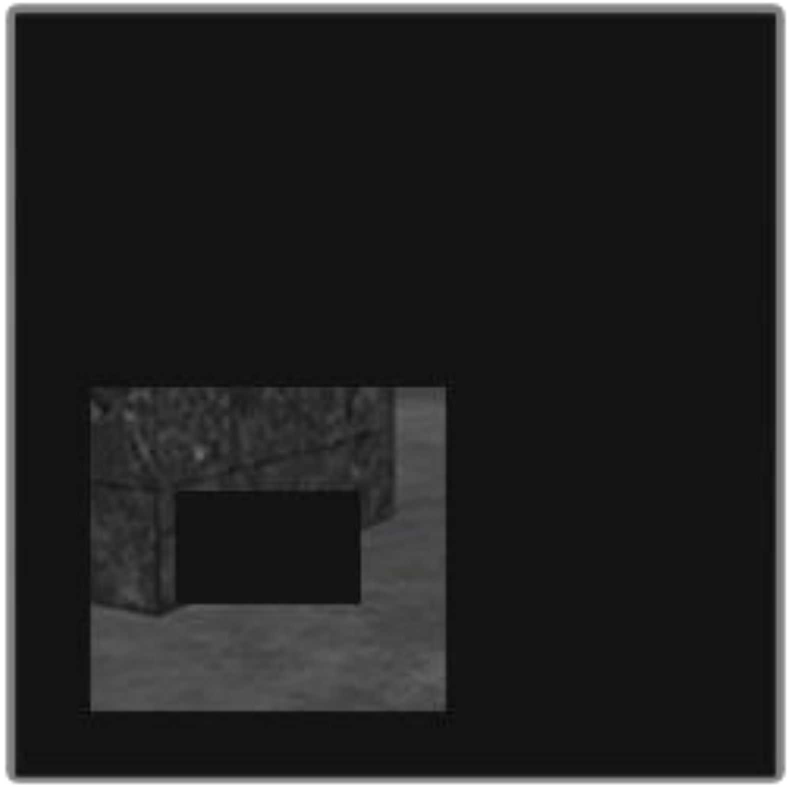
Source: https://codepen.io/anon/pen/EwrbGZ


Color buffer



```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0
0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Stencil buffer


After stencil test

Source: https://learnopengl.com/Advanced-OpenGL/Stencil-testing

3. Intro to Stencil test

- It allows selecting a part of the rendering window, but based on a pre-defined texture, or other user input

3. Intro to Stencil test (example)

Source: https://learnopengl.com/Advanced-OpenGL/Stencil-testing

3. Intro to Stencil test (example)

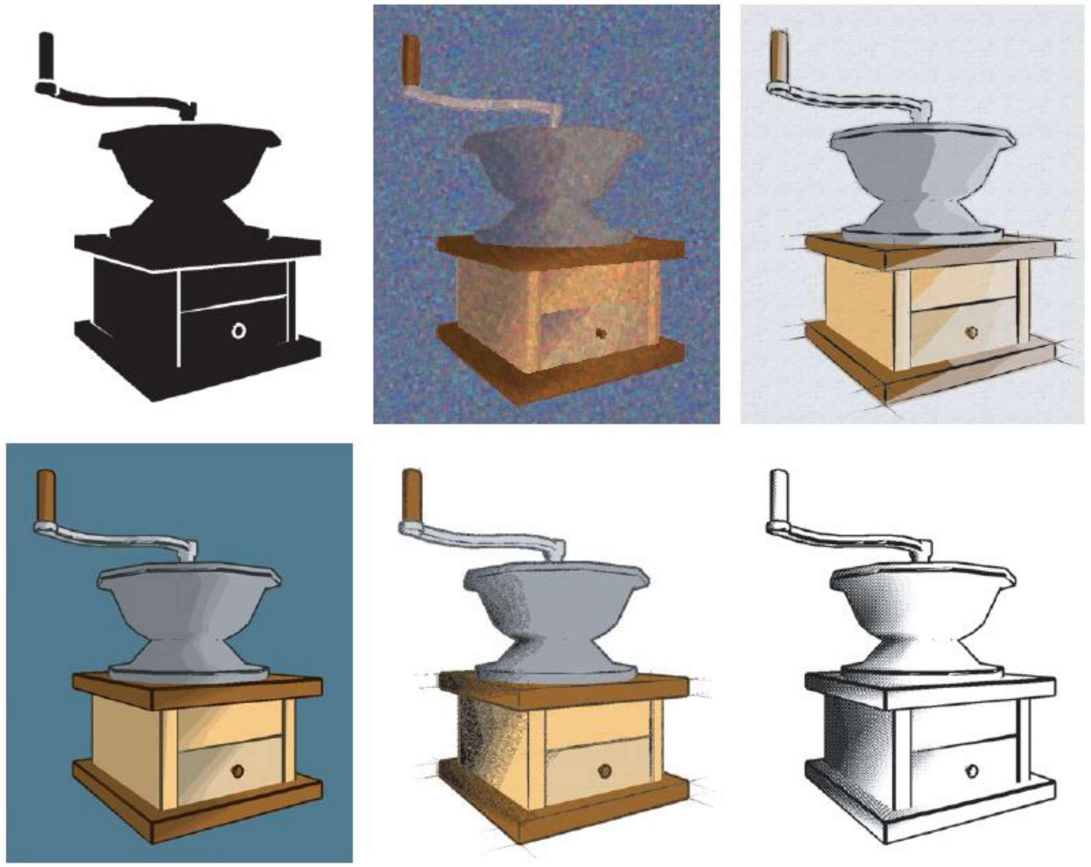
```
glEnable(GL_STENCIL_TEST);


// Draw floor
glStencilFunc(GL_ALWAYS, 1, 0xFF); // Set any stencil to 1
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
glStencilMask(0xFF); // Write to stencil buffer
glDepthMask(GL_FALSE); // Don't write to depth buffer
glClear(GL_STENCIL_BUFFER_BIT); // Clear stencil buffer (0 by default)
glDrawArrays(GL_TRIANGLES, 36, 6); // Draw cube reflection
glStencilFunc(GL_EQUAL, 1, 0xFF); // Pass test if stencil value is 1
glStencilMask(0x00); // Don't write anything to stencil buffer
glDepthMask(GL_TRUE); // Write to depth buffer
model = glm::scale( glm::translate(model, glm::vec3(0, 0, -1)),
glm::vec3(1, 1, -1) );
glUniformMatrix4fv(uniModel, 1, GL_FALSE, glm::value_ptr(model));
glDrawArrays(GL_TRIANGLES, 0, 36); glDisable(GL_STENCIL_TEST);
```

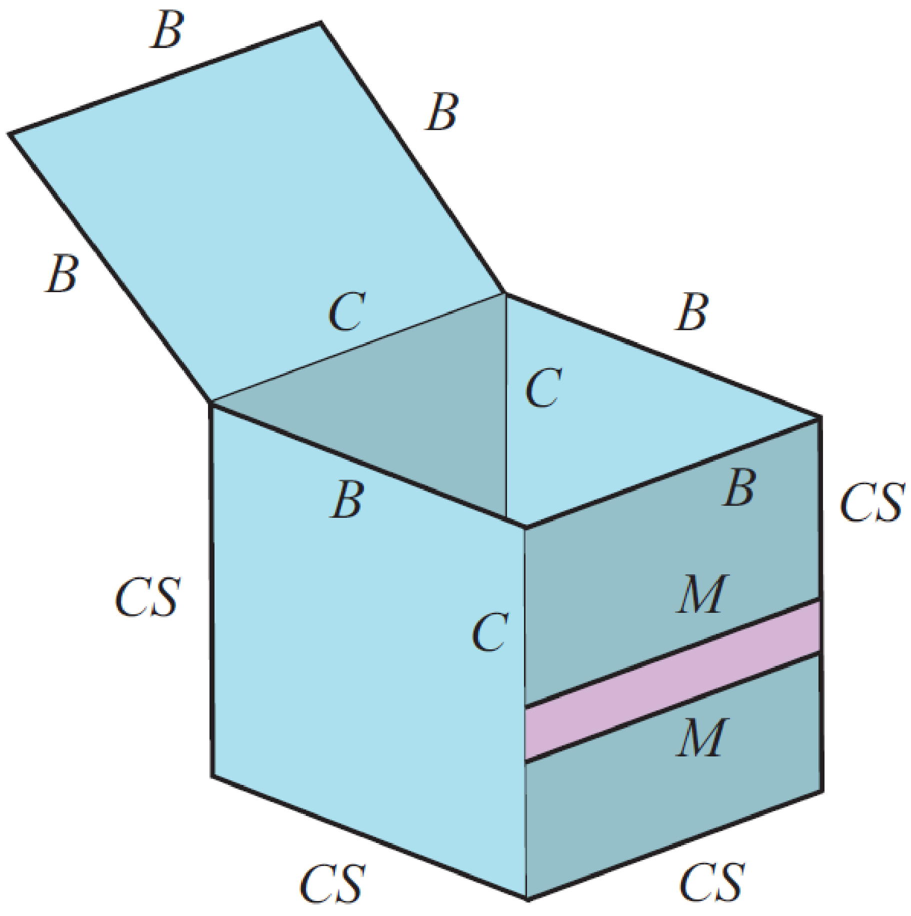
ESCOLA DE NOVES TECNOLOGIES INTERACTIVES

UNIVERSITAT DE BARCELONA

Source: https://learnopengl.com/Advanced-OpenGL/Stencil-testing

4. Intro to contour rendering





4. Intro to contour rendering

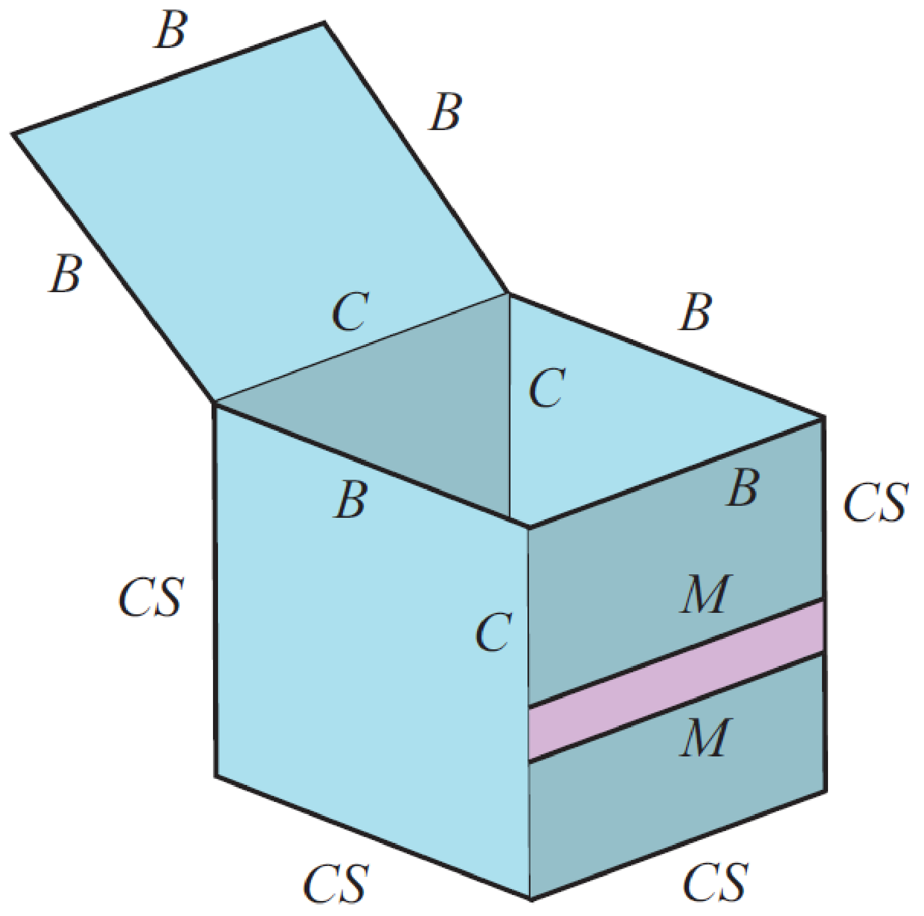
We have:

B: boundary

C: crease

M: material

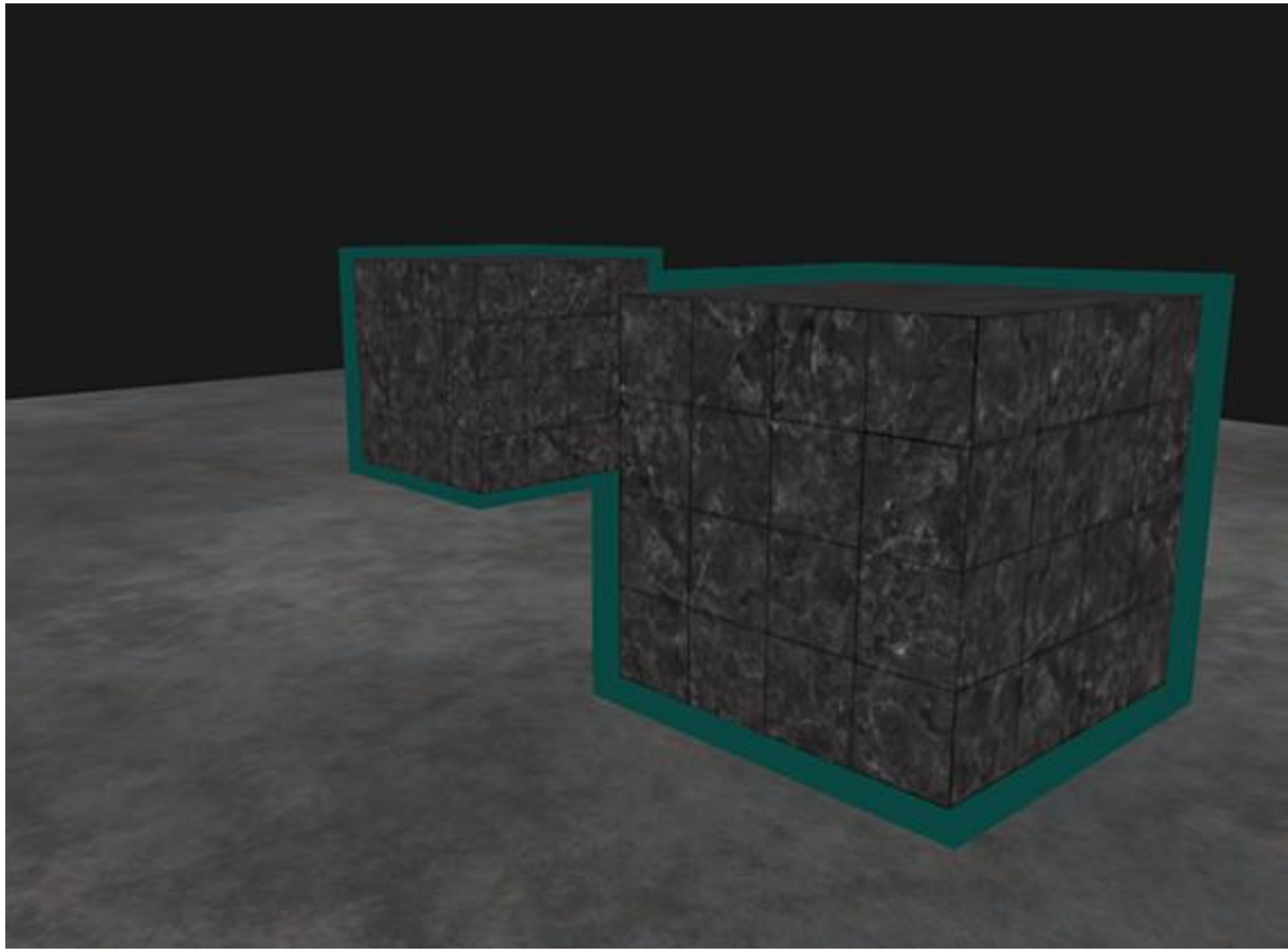
S: silhouette



Draw Silhouette with Stencil Buffer

How to use a stencil buffer for contour rendering:

1. Set the stencil func to GL_ALWAYS before drawing the (to be outlined) objects, updating the stencil buffer with 1s wherever the objects' fragments are rendered.
2. Render the objects.
3. Disable stencil writing and depth testing.
4. Scale each of the objects by a small amount.
5. Use a different fragment shader that outputs a single (border) color.
6. Draw the objects again, but only if their fragments' stencil values are not equal to 1.
7. Enable stencil writing and depth testing again.



See: https://learnopengl.com/Advanced-OpenGL/Stencil-testing

Draw Silhouette with Stencil Buffer

Benefits:

- Simple
- Well integrated in the graphics pipeline

Possible Drawbacks:

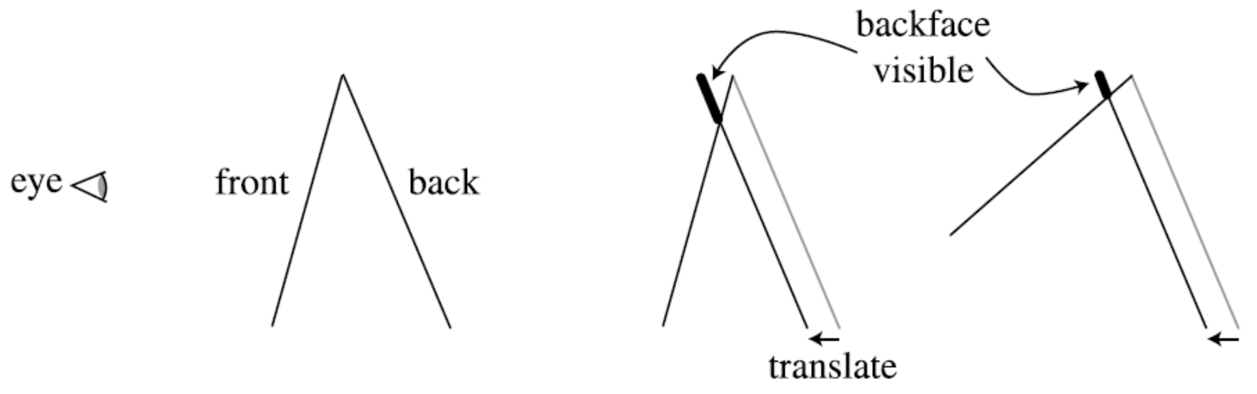
- No Crease rendered
- No Boundary rendered

Other options:

- Detect changes in depth buffer (Silhouette, but also Border)
- Detect changes in normal directions (> 45º → Crease)

Other rendering methods:

- Draw Lines in openGL
- Generate geometry for contours in geometry buffer
- Grow triangles
- Expand and render backfaces in black
- Move and render backfaces in black



Resources

See online references indicated in the previous slides

Also:

- [Kessenich] Kessenich et al. OpenGL Programming Guide. Chapter 7. Light and Shadow
- [Akenine-Möller] Akenine-Möller et al. Real-Time Rendering. Third Edition, CRC Press (chapter 11)
- https://learnopengl.com/Lighting/Basic-Lighting
- https://en.wikipedia.org/wiki/Phong_reflection_model