02561 Computer Graphics

3D models and asynchronous data loading

Jeppe Revall Frisvad

October 2021

Acquiring 3D models

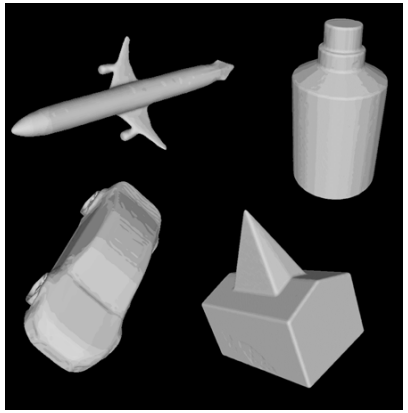

- Stanford 3D scanning repository
  https://graphics.stanford.edu/data/3Dscanrep/



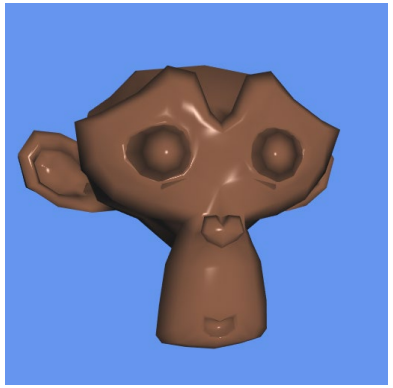
- McGuire computer graphics archive
  https://casual-effects.com/data/

- Thingiverse
  https://www.thingiverse.com/

- ShapeNet
  https://www.shapenet.org/

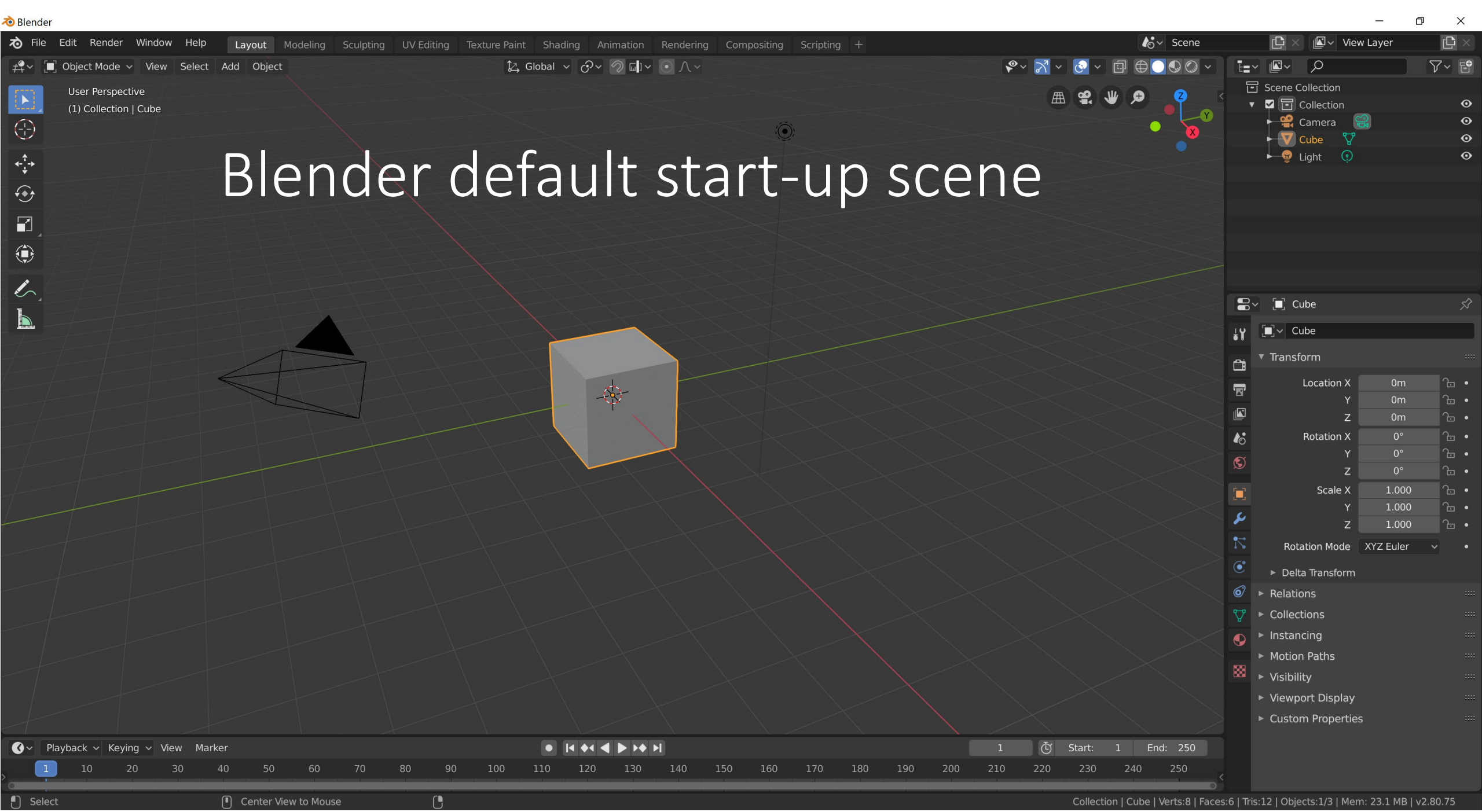


- Modeling tools:
  - Maya https://www.autodesk.com/products/maya/
  - Blender https://www.blender.org/
  - …

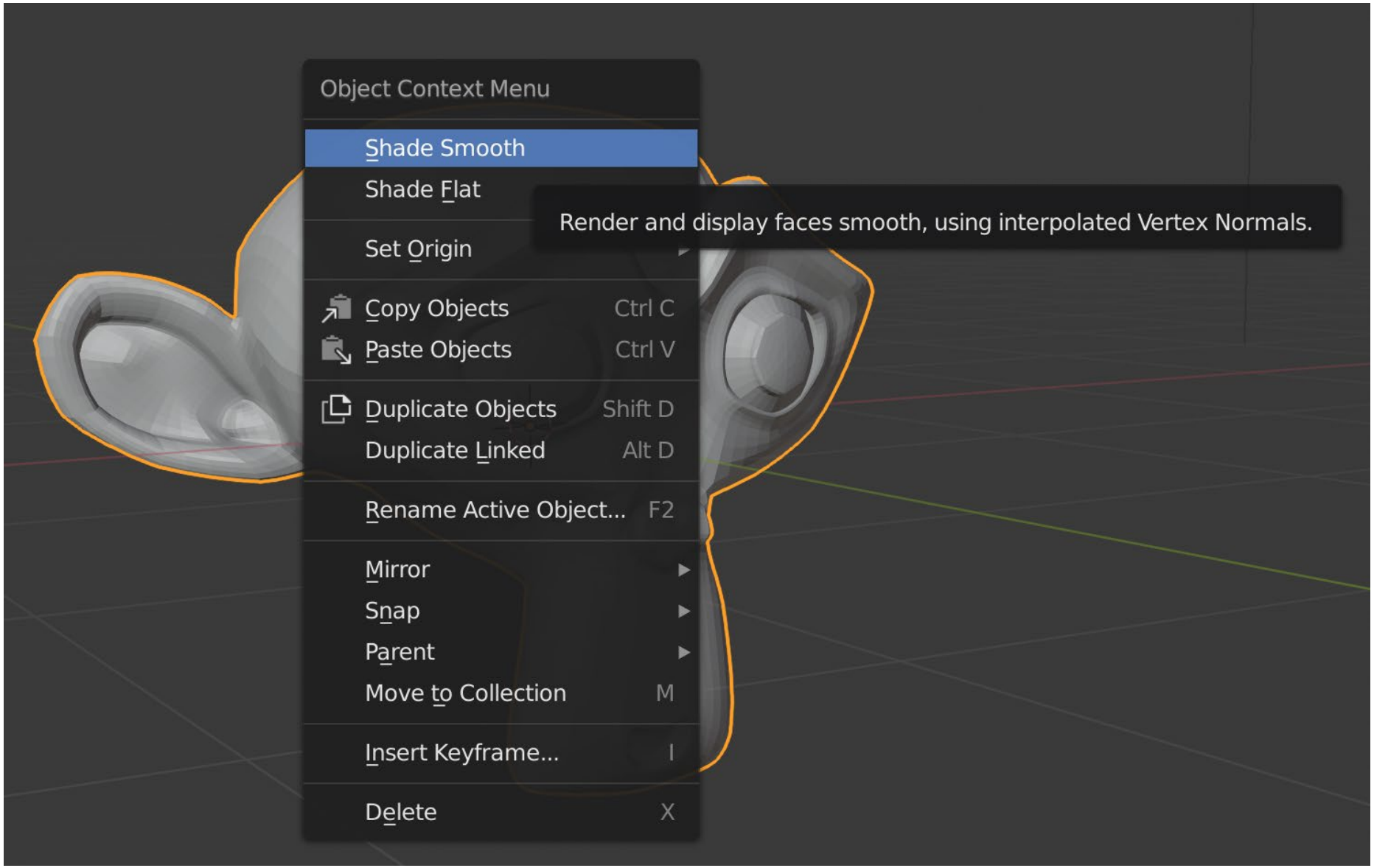


Export from Blender

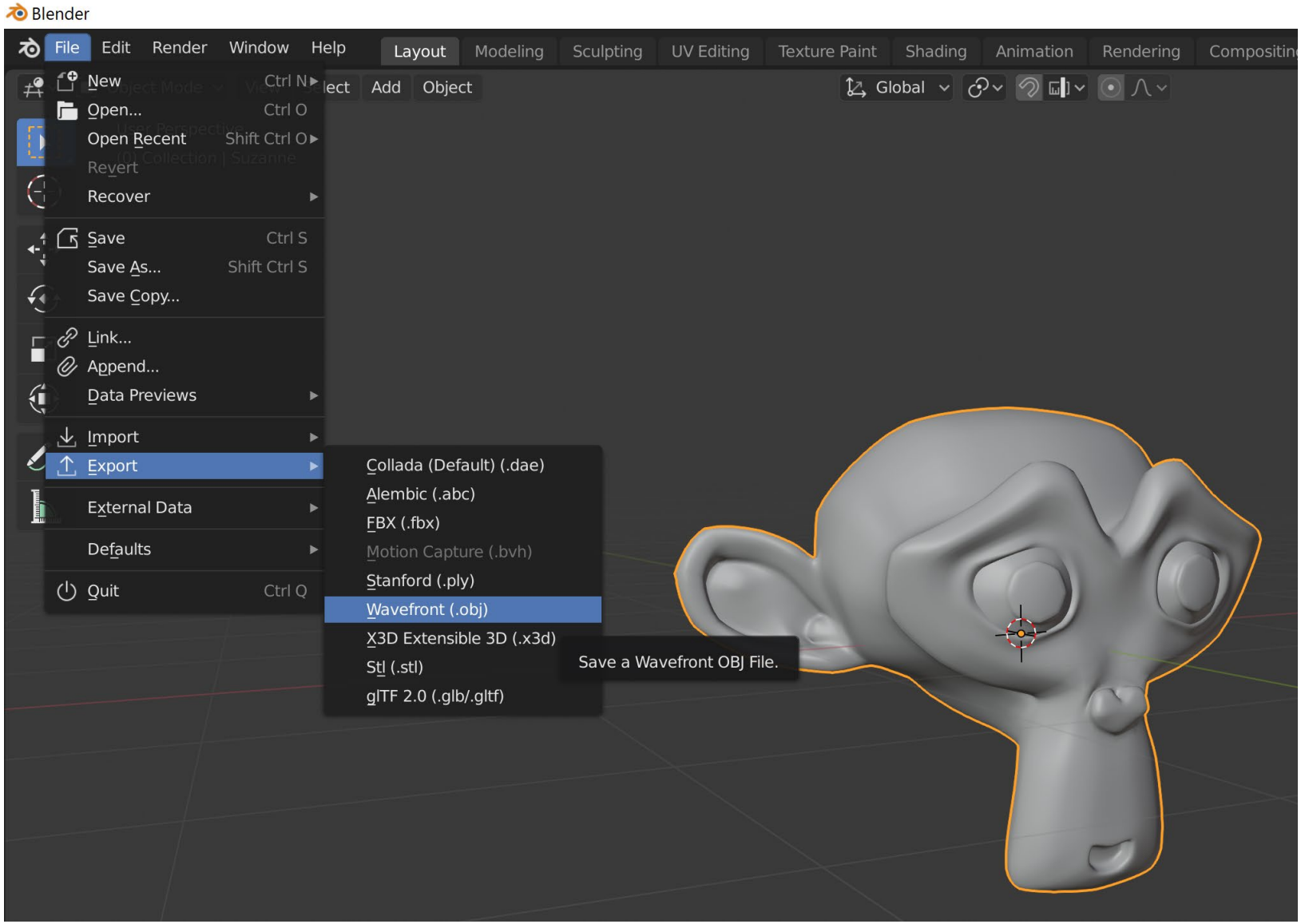
1. Click the splash screen and right click and Delete the default cube.
2. Create a model (start from the Add $\rightarrow$ Mesh menu, for example).
3. Right click and select Shade Smooth to get interpolated vertex normals.
4. Export mesh: File $\rightarrow$ Export $\rightarrow$ Wavefront (.obj)
5. Select export options (deselect Include UVs, for example).
6. Choose folder and .obj file name and press Export OBJ.

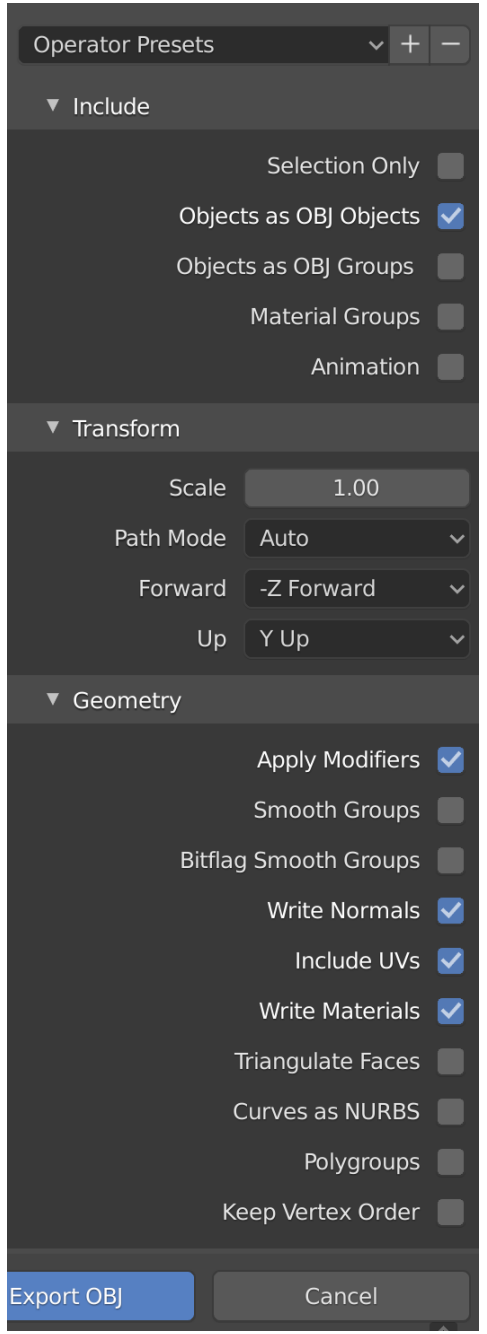

Blender
File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Scripting
Scene
View Layer
Object Mode View Select Add Object
Global
User Perspective
(1) Collection | Cube
Blender default start-up scene
Scene Collection
Collection
Camera
Cube
Light
Cube
Cube
Transform
Location X 0m
Y 0m
Z 0m
Rotation X 0°
Y 0°
Z 0°
Scale X 1.000
Y 1.000
Z 1.000
Rotation Mode XYZ Euler
Delta Transform
Relations
Collections
Instancing
Motion Paths
Visibility
Viewport Display
Custom Properties
Playback Keying View Marker
1
Start: 1
End: 250
1 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250
Select
Center View to Mouse
Collection | Cube | Verts:8 | Faces:6 | Tris:12 | Objects:1/3 | Mem: 23.1 MB | v2.80.75

Right click to get the Object Context Menu



Blender export to Wavefront OBJ





Wavefront OBJ (.obj and .mtl files)

cube.obj - Notepad 2e x64

File Edit View Settings ?

```
1# Blender v2.80 (sub 75) OBJ File: ''
2# www.blender.org
3mtllib cube.mtl
4o Cube
5v 1.000000 1.000000 -1.000000
6v 1.000000 -1.000000 -1.000000
7v 1.000000 1.000000 1.000000
8v 1.000000 -1.000000 1.000000
9v -1.000000 1.000000 -1.000000
10v -1.000000 -1.000000 -1.000000
11v -1.000000 1.000000 1.000000
12v -1.000000 -1.000000 1.000000
13vn 0.0000 1.0000 0.0000
14vn 0.0000 0.0000 1.0000
15vn -1.0000 0.0000 0.0000
16vn 0.0000 -1.0000 0.0000
17vn 1.0000 0.0000 0.0000
18vn 0.0000 0.0000 -1.0000
19usemtl Material
20s off
21f 1//1 5//1 7//1 3//1
22f 4//2 3//2 7//2 8//2
23f 8//3 7//3 5//3 6//3
24f 6//4 2//4 4//4 8//4
25f 2//5 1//5 3//5 4//5
26f 6//6 5//6 1//6 2//6
27
```

cube.mtl - Notepad 2e x64

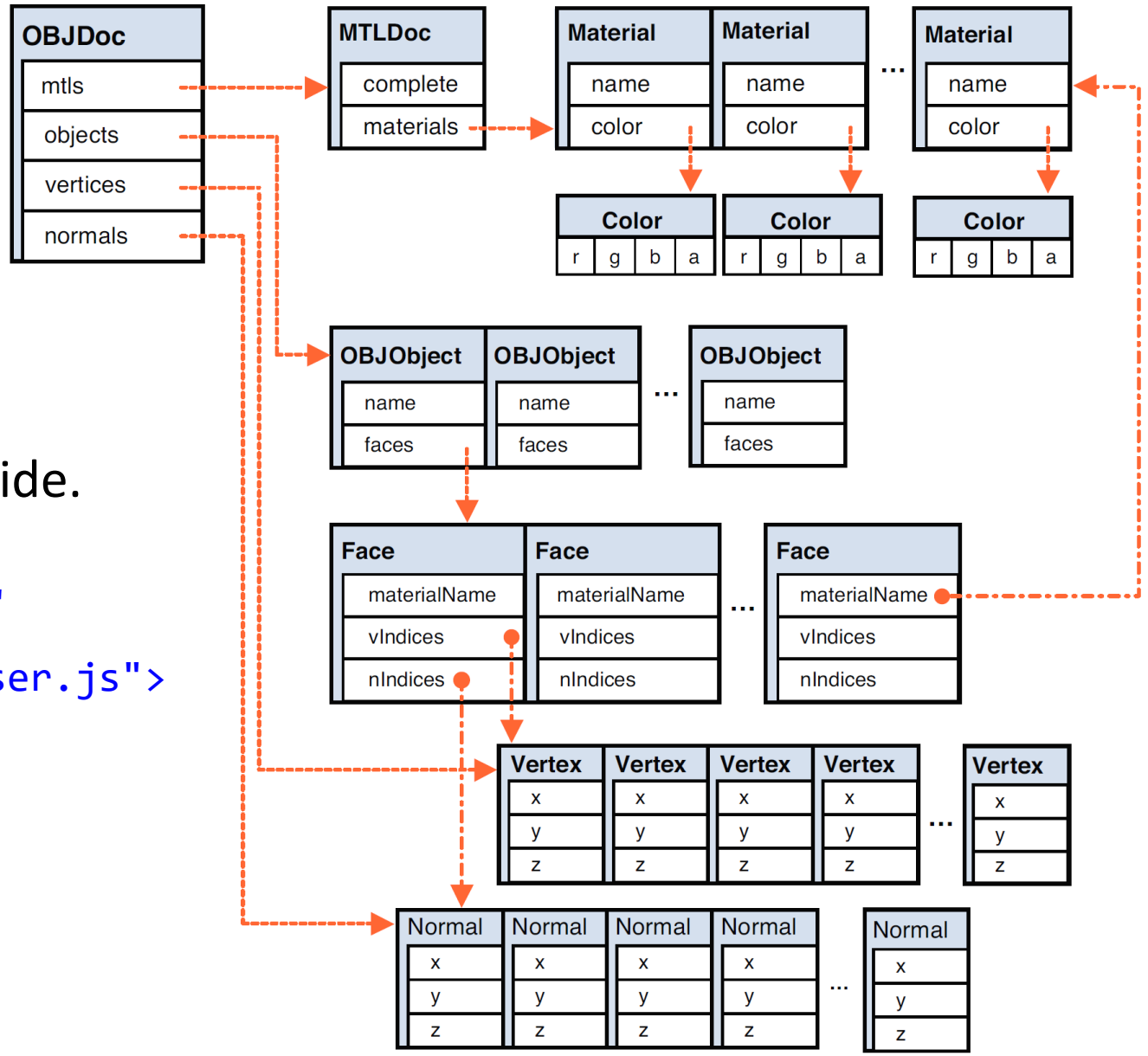
File Edit View Settings ?

```
1# Blender MTL File: 'None'
2# Material Count: 1
3
4newmtl Material
5Ns 323.999994
6Ka 1.000000 1.000000 1.000000
7Kd 0.800000 0.800000 0.800000
8Ks 0.500000 0.500000 0.500000
9Ke 0.0 0.0 0.0
10Ni 1.450000
11d 1.000000
12illum 2
13
```

Parsing Wavefront OBJ files

Implemented in **OBJParser.js**
from WebGL Programming Guide.

```
<script type="text/javascript"
        src="../common/OBJParser.js">
</script>
```

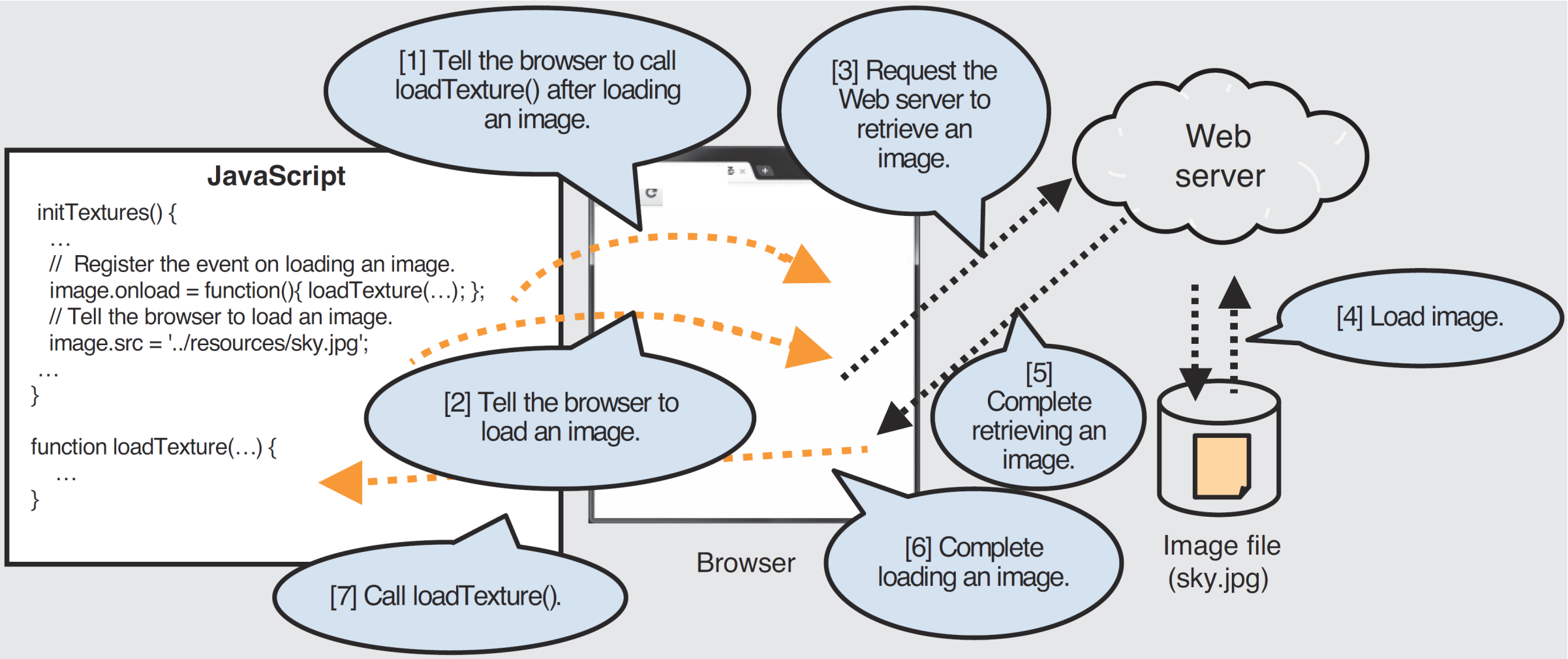


Loading an OBJ file into WebGL

- Suppose shaders have been loaded to gl.program and have 3 attribute variables:
  - a_Position
  - a_Normal
  - a_Color
- Let us implement a function for loading and initializing a mesh object.
- We need some helper functions from WebGL Programming Guide.

```
function initObject(gl, obj_filename, scale)
{
  gl.program.a_Position = gl.getAttribLocation(gl.program, 'a_Position');
  gl.program.a_Normal = gl.getAttribLocation(gl.program, 'a_Normal');
  gl.program.a_Color = gl.getAttribLocation(gl.program, 'a_Color');

  // Prepare empty buffer objects for vertex coordinates, colors, and normals
  var model = initVertexBuffers(gl);

  // Start reading the OBJ file
  readOBJFile(obj_filename, gl, model, scale, true);

  return model;
}


// Create a buffer object and perform the initial configuration
function initVertexBuffers(gl) { ... }
function createEmptyArrayBuffer(gl, a_attribute, num, type) { ... }

var g_objDoc = null;      // Info parsed from OBJ file
var g_drawingInfo = null; // Info for drawing the 3D model with WebGL

// Asynchronous file loading (request, parse, send to GPU buffers)
function readOBJFile(fileName, gl, model, scale, reverse) { ... }
function onReadOBJFile(fileString, fileName, gl, o, scale, reverse) { ... }
function onReadComplete(gl, model, objDoc) { ... }
```

Asynchronous data loading



When loading an OBJ file:

`onReadOBJFile` corresponds to `image.onload`

`image.src` becomes an http request to open a file

`onReadComplete` corresponds to `loadTexture`

Rendering a loaded object

- Use an `animate` function to call the render function repeatedly.
- Wait for parsed data (`g_objDoc`) to be available then initialize WebGL buffers (`onReadComplete`).
- Once WebGL buffers are available (`g_drawingInfo`) draw the loaded mesh as an indexed face set.

```
function render(gl, view, model)
{
  if (!g_drawingInfo && g_objDoc && g_objDoc.isMTLComplete()) {
    // OBJ and all MTLs are available
    g_drawingInfo = onReadComplete(gl, model, g_objDoc);
  }
  if (!g_drawingInfo) return;

  ⋮

  gl.drawElements(gl.TRIANGLES, g_drawingInfo.indices.length, gl.UNSIGNED_SHORT, 0);
}
```

File access from files

- Local file access is restricted to maintain browser security.
- To work locally, use Python to set up a local server:
  - Open a command prompt
  - Go to folder with your solution files and library files (possibly in subfolders).
  - Write the command (might vary slightly on different systems):

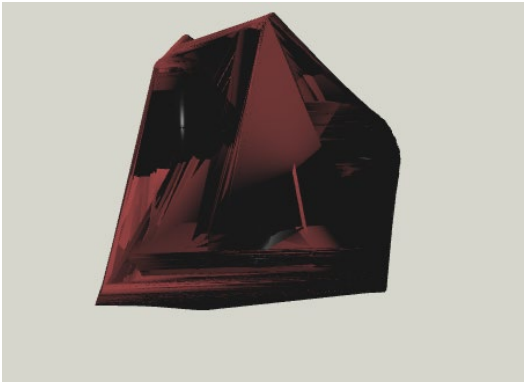
```
python -m http.server
```

- Alternative: Upload your webpage to a server and run your program by visiting the webpage (then you are no longer working locally).
- You have a student webspace http://www.student.dtu.dk/~username/ http://gbar.dtu.dk/faq/50-homepage
- Upload files using SCP or SFTP (we recommend WinSCP for Windows).

Extension: high poly count objects

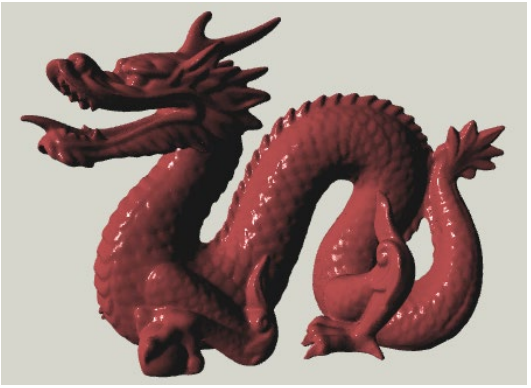
- Let us try to render a high poly object.
- The Stanford dragon is ~560k vertices, ~1100k triangles.
- When trying, it seems to be inside-out.
- We need an extension: OES_element_index_uint
  https://www.khronos.org/registry/webgl/extensions/

```
var ext = gl.getExtension('OES_element_index_uint');
if (!ext) {
  console.log('Warning: Unable to use an extension');
}
```



- This enables use of gl.UNSIGNED_INT (32 bits) instead of gl.UNSIGNED_SHORT (16 bits) when rendering an indexed face set. Modify one line in ObjParser.js:

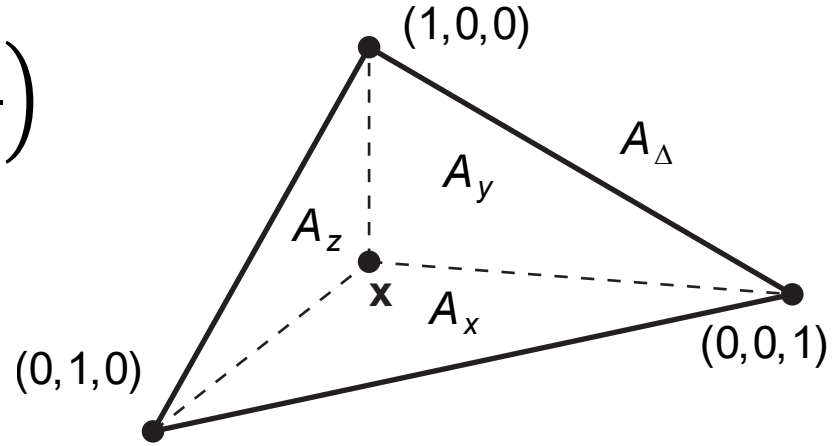
```
var indices = new Uint32Array(numIndices);
```



- Then draw using: `gl.drawElements(gl.TRIANGLES, g_drawingInfo.indices.length, gl.UNSIGNED_INT, 0);`

Extension: adding an attribute

- Sometimes it is useful to be able to add another vertex attribute.
- This can be done in the parser functions `getDrawingInfo` and `DrawingInfo`.
- Example 1:
  - For triangle vertices $\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2$, we could add the corresponding attributes $(1,0,0), (0,1,0), (0,0,1)$.
  - Assigning this attribute to a varying variable in the vertex shader, it would become barycentric coordinates in the fragment shader.
- Barycentric coordinates: $(\alpha, \beta, \gamma) = \left(\frac{A_x}{A_\Delta}, \frac{A_y}{A_\Delta}, \frac{A_z}{A_\Delta}\right)$
- $\boldsymbol{x} = \alpha\boldsymbol{v}_0 + \beta\boldsymbol{v}_1 + \gamma\boldsymbol{v}_2$
- These are useful for wireframe rendering among many other things.

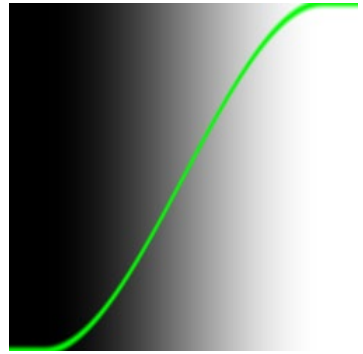


- We could also add the Bézier control point attribute from Week 3.

Wireframe rendering

- Given barycentric coordinates (`bc`) of a point $\boldsymbol{x}$ in a triangle, the point is close to an edge if a barycentric coordinate is close to 0.
- The rate of change in bc between neighboring pixels scales with the distance to the camera. We can get this rate using an extension.
- Using the rate of change (`fwidth(bc)`) and the smoothstep function, we can make a distance invariant wireframe rendering.

```
#extension GL_OES_standard_derivatives : enable
float edge_factor(vec3 bc){
    vec3 d = fwidth(bc);
    vec3 a3 = smoothstep(vec3(0.0), d*u_LineWidth, bc);
    return u_LineWidth > 0.0 ? min(min(a3.x, a3.y), a3.z) : 1.0;
}
```



smoothstep

- The edge factor is for linear interpolation between wireframe color and the color computed by the shader.