UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

CG Basics V
Loading Models in WebGL

Bachelor's Degree in Video Game Design and Development

Drawing Geometry

- In order to draw geometric models we must specify
  - Vertices coordinates
  - Colors
  - Normals
  - Texture coordinates
  - …
- This is done by allocating buffers on GPU and storing there the information of the vertices (*loadSceneOnGPU()*)

Drawing Geometry

- **loadSceneOnGPU()**

```
var squareVertexPositionBuffer;
var squareVertexColorBuffer;

function loadSceneOnGPU() {
    squareVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);
    vertices = [
         1.0,  1.0,  0.0,
        -1.0,  1.0,  0.0,
         1.0, -1.0,  0.0,
        -1.0, -1.0,  0.0
    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
    squareVertexPositionBuffer.itemSize = 3;
    squareVertexPositionBuffer.numItems = 4;

    squareVertexColorBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);
    colors = [
         1.0, 1.0, 0.0, 1.0,
         1.0, 1.0, 0.0, 1.0,
         1.0, 1.0, 0.0, 1.0,
         1.0, 1.0, 0.0, 1.0
    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(colors), gl.STATIC_DRAW);
    squareVertexColorBuffer.itemSize = 4;
    squareVertexColorBuffer.numItems = 4;
}
```

Create the buffer to store the vertex information

Bind it as an array buffer to set it “active”

Load the information on the GPU buffer

Drawing Geometry

- **The information stored in the buffers (array buffer) are used to draw primitives (*drawScene()*)**

```
...

gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexPositionBuffer);

gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
squareVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

gl.bindBuffer(gl.ARRAY_BUFFER, squareVertexColorBuffer);

gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
squareVertexColorBuffer.itemSize, gl.FLOAT, false, 0, 0);

sendMatricesToShader();
gl.drawArrays(gl.TRIANGLE_STRIP, 0, squareVertexPositionBuffer.numItems);

...
```

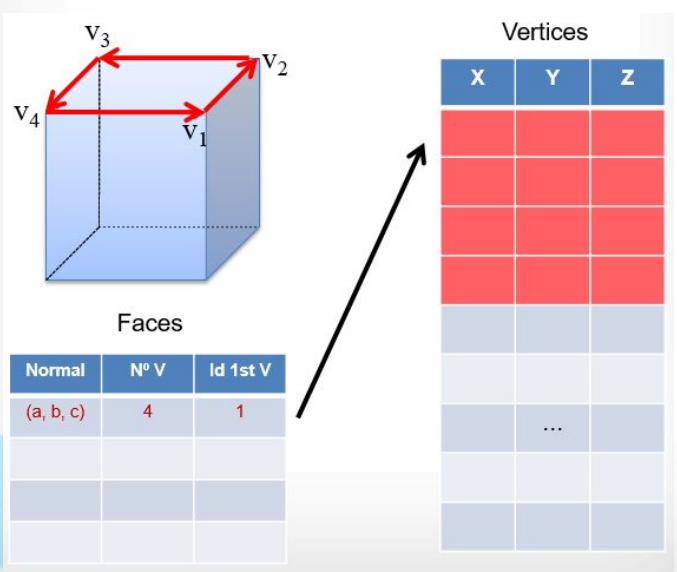
Set the buffers with the information “active”

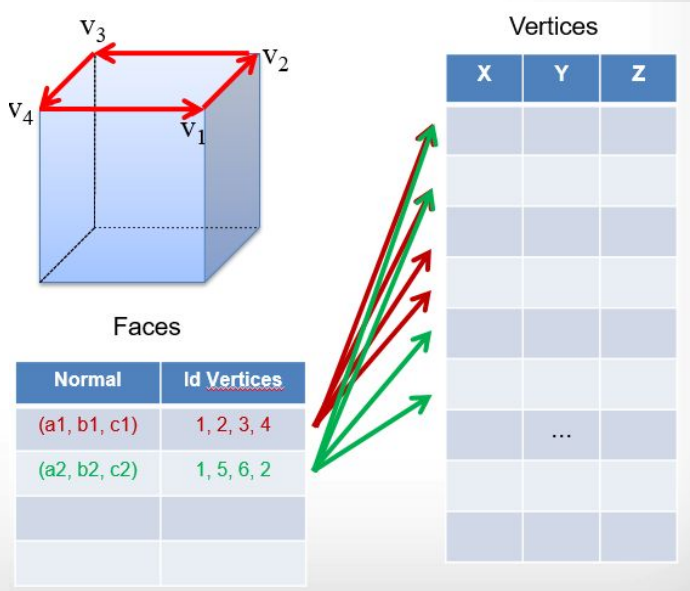
Send the information to the shaders

Draw the information as a triangle strips

Drawing Geometry

- **Vertex information can be defined once for each primitive containing the vertex, or it can be defined once and reused for different primitives**





Drawing Geometry

- To avoid repeating vertex information, the vertices of each primitive must be specified by using an ELEMENT_ARRAY_BUFFER (*loadSceneOnGPU()*)

```
cubeVertexPositionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
var vertices = [
    // Front face
    -1.0, -1.0,  1.0,
     1.0, -1.0,  1.0,
     1.0,  1.0,  1.0,
    -1.0,  1.0,  1.0,

    ...

    // Left face
    -1.0, -1.0, -1.0,
    -1.0, -1.0,  1.0,
    -1.0,  1.0,  1.0,
    -1.0,  1.0, -1.0
];
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
cubeVertexPositionBuffer.itemSize = 3;
cubeVertexPositionBuffer.numItems = 24;
```

```
cubeVertexIndexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
var cubeVertexIndices = [
    0, 1, 2,      0, 2, 3,    // Front face
    4, 5, 6,      4, 6, 7,    // Back face
    8, 9, 10,     8, 10, 11,  // Top face
    12, 13, 14,   12, 14, 15, // Bottom face
    16, 17, 18,   16, 18, 19, // Right face
    20, 21, 22,   20, 22, 23  // Left face
];
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(cubeVertexIndices),
              gl.STATIC_DRAW);
cubeVertexIndexBuffer.itemSize = 1;
cubeVertexIndexBuffer.numItems = 36;
```

Drawing Geometry

- **Then, this buffer is used to render the primitives (*drawScene()*)**

```
function drawScene() {
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(45, gl.viewportWidth / gl.viewportHeight, 0.1, 100.0, pMatrix);

    mat4.identity(mvMatrix);
    mat4.translate(mvMatrix, [0.0, 1.5, -10.0]);

    gl.bindBuffer(gl.ARRAY_BUFFER, cubeVertexPositionBuffer);
    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
                           cubeVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, cubeVertexIndexBuffer);
    sendMatricesToShader();
    gl.drawElements(gl.TRIANGLES, cubeVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);

}
```

Loading Models

- Models in WebGL normally use the *JSON* format (javascript objects)

- JSON 3d models basically contain javascript lists with the information of the vertices to load on GPU and some metadata

```
{
    "vertexPositions" : [5.929688,4.125,0,5.387188,4.125,2.7475,...],
    "vertexNormals" : [-0.966742,-0.255752,0,-0.893014,-0.256345,-0.369882,...],
    "vertexTextureCoords" : [2,2,1.75,2,1.75,1.975,...],
    "indices" : [0,1,2,2,3,0,...]
}
```

Loading Models

- To load JSON models we must
  - retrieve the information of the vertices

```
function loadModel() {
    var request = new XMLHttpRequest();
    request.open("GET", "model.json");
    request.onreadystatechange = function () {
        if (request.readyState == 4) {
            handleLoadedModel(JSON.parse(request.responseText));
        }
    }
    request.send();
}
```

Loading Models

- To load JSON models we must
  - Store vertices' information on GPU buffers

```
var modelVertexPositionBuffer;
var modelVertexNormalBuffer;
var modelVertexTextureCoordBuffer;
var modelVertexIndexBuffer;

function handleLoadedModel(modelData) {
    modelVertexNormalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, modelVertexNormalBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(modelData.vertexNormals), gl.STATIC_DRAW);
    modelVertexNormalBuffer.itemSize = 3;
    modelVertexNormalBuffer.numItems = modelData.vertexNormals.length / 3;

    modelVertexTextureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, modelVertexTextureCoordBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(modelData.vertexTextureCoords), gl.STATIC_DRAW);
    modelVertexTextureCoordBuffer.itemSize = 2;
    modelVertexTextureCoordBuffer.numItems = modelData.vertexTextureCoords.length / 2;

    modelVertexPositionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, modelVertexPositionBuffer);
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(modelData.vertexPositions), gl.STATIC_DRAW);
    modelVertexPositionBuffer.itemSize = 3;
    modelVertexPositionBuffer.numItems = modelData.vertexPositions.length / 3;

    modelVertexIndexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, modelVertexIndexBuffer);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(modelData.indices), gl.STATIC_DRAW);
    modelVertexIndexBuffer.itemSize = 1;
    modelVertexIndexBuffer.numItems = modelData.indices.length;

}
```

Loading Models

- To load JSON models we must
  - Draw the model by using the allocated buffers (*drawScene()*)

```
function drawScene() {
    gl.viewport(0, 0, gl.viewportWidth, gl.viewportHeight);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    mat4.perspective(fov, gl.viewportWidth / gl.viewportHeight, 0.1, 300.0, pMatrix);
    //mat4.ortho(-10.0, 10.0, -2.0, 2.0, 0.1, 100.0, pMatrix);

    ...

    if (modelVertexPositionBuffer)
    {
        gl.bindBuffer(gl.ARRAY_BUFFER, modelVertexPositionBuffer);
        gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
                               modelVertexPositionBuffer.itemSize, gl.FLOAT, false, 0, 0);
        gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, modelVertexIndexBuffer);
        sendMatricesToShader();
        gl.drawElements(gl.TRIANGLES, modelVertexIndexBuffer.numItems, gl.UNSIGNED_SHORT, 0);
    }
}
```

Loading Models

- 3D models can be stored in many different formats
  - Obj, fbx, 3ds, blend, …
- There are converters to export these models to JSON
- Unfortunately, some errors while exporting from other formats may appear
  - Vertices assigned to incorrect faces
  - Normals in opposite directions
  - Incorrect texture coordinates…
- Three.js is a library based on WebGL that can be used to import directly other formats to our applications

Questions?

www.citm.upc.edu