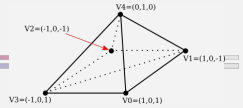
Modeling – Polymeshes

- object type – `THREE.Mesh`
  - all faces are triangles
- geometry type – `THREE.BufferGeometry`
  - note that this is correct in the code examples in the book but not in the text itself!
  - order according to the counterclockwise-from-the-outside convention

Modeling – Polymeshes




```
let pyramidVertices = new Float32Array( [
              // Data for the pyramidGeom "position" attribute.
              // Contains the x,y,z coordinates for the vertices.
              // Each group of three numbers is a vertex;
              // each group of three vertices is one face.
            -1,0,1,  -1,0,-1,  1,0,-1, // First triangle in the base.
            -1,0,1,   1,0,-1,  1,0,1,  // Second triangle in the base.
            -1,0,1,   1,0,1,   0,1,0,  // Front face.
             1,0,1,   1,0,-1,  0,1,0,  // Right face.
             1,0,-1, -1,0,-1,  0,1,0,  // Back face.
            -1,0,-1, -1,0,1,   0,1,0   // Left face.
        ] );
let pyramidGeom = new THREE.BufferGeometry();
pyramidGeom.setAttribute("position",
              new THREE.BufferAttribute(pyramidVertices,3) );
```

```
pyramidVertices = new Float32Array( [
             1, 0,  1,  // vertex number 0
             1, 0, -1,  // vertex number 1
            -1, 0, -1,  // vertex number 2
            -1, 0,  1,  // vertex number 3
             0, 1,  0   // vertex number 4
   ] );

pyramidFaceIndexArray = [
            3, 2, 1,  // First triangle in the base.
            3, 1, 0,  // Second Triangle in the base.
            3, 0, 4,  // Front face.
            0, 1, 4,  // Right face.
            1, 2, 4,  // Back face.
            2, 3, 4   // Left face.
   ];

pyramidGeom = new THREE.BufferGeometry();
pyramidGeom.setAttribute("position",
              new THREE.BufferAttribute(pyramidVertices,3) );
pyramidGeom.setIndex( pyramidFaceIndexArray );
```

- basic representation – no face indices
  - suitable for a polyhedron
- index face set representation
  - a vertex index references all of the data for the vertex – coordinates, normal, color, etc
  - suitable when mesh approximates a smooth surface
- set a material to have the same color for all faces

```
pyramid = new THREE.Mesh(
              pyramidGeom,
              new THREE.MeshLambertMaterial({ color: "yellow" })
            );
```


Modeling – Polymeshes

- lighting (`MeshLambertMaterial`, `MeshPhongMaterial`) requires normals
  - flat shading only needs face normals
  - smooth shading requires vertex normals
- *geom*`.computeVertexNormals();`
  - for a basic polymesh, computes face normals
    - each vertex belongs to one face and gets the normal for that face
    - to use smooth shading, manually set the `normal` attribute
  - for an indexed face set, computes vertex normals
    - vertices are not duplicated, and normal is set to the average of the face normals the vertex belongs to
    - to handle creases on an otherwise smooth surface, duplicate those vertices in the geometry

Modeling – Polymeshes

- there are multiple options for assigning colors/materials to polymeshes
  - one material for the entire mesh

```
pyramid = new THREE.Mesh(
              pyramidGeom,
              new THREE.MeshLambertMaterial({ color: "yellow" })
            );
```

  - different materials for each face
  - different colors for each vertex

Modeling – Polymeshes

for the basic representation

- different materials can be assigned to different faces
  - define groups of vertices within the `BufferGeometry` and assign a material index to each
  - specify an array of materials when creating the mesh

```
let pyramidVertices = new Float32Array( [
            // Data for the pyramidGeom "position" attribute.
            // Contains the x,y,z coordinates for the vertices.
            // Each group of three numbers is a vertex;
            // each group of three vertices is one face.
          -1,0,1,  -1,0,-1,  1,0,-1, // First triangle in the base.
          -1,0,1,   1,0,-1,  1,0,1,  // Second triangle in the base.
          -1,0,1,   1,0,1,   0,1,0,  // Front face.
           1,0,1,   1,0,-1,  0,1,0,  // Right face.
           1,0,-1, -1,0,-1,  0,1,0,  // Back face.
          -1,0,-1, -1,0,1,   0,1,0   // Left face.
        ] );
let pyramidGeom = new THREE.BufferGeometry();
pyramidGeom.setAttribute("position",
              new THREE.BufferAttribute(pyramidVertices,3) );
```

first vertex in group (index in vertex array / 3)

number of vertices in group (number of faces in group x 3)

index in material array

```
pyramidGeom.addGroup(0,6,0);  // The base (2 triangles)
pyramidGeom.addGroup(6,3,1);  // Front face.
pyramidGeom.addGroup(9,3,2);  // Right face.
pyramidGeom.addGroup(12,3,3); // Back face.
pyramidGeom.addGroup(15,3,4); // Left face.
```

```
pyramidMaterialArray= [
          // Array of materials, for use as pyramids's material.
        new THREE.MeshLambertMaterial( { color: 0xffffff } ),
        new THREE.MeshLambertMaterial( { color: 0x99ffff } ),
        new THREE.MeshLambertMaterial( { color: 0xff99ff } ),
        new THREE.MeshLambertMaterial( { color: 0xffff99 } ),
        new THREE.MeshLambertMaterial( { color: 0xff9999 } )
      ];
```

```
pyramid = new THREE.Mesh( pyramidGeom, pyramidMaterialArray );
```


Modeling – Polymeshes

for the indexed face set representation

- different materials can be assigned to different faces
  - define groups of vertices within the `BufferGeometry` and assign a material index to each
  - specify an array of materials when creating the mesh

```
pyramidVertices = new Float32Array( [
            1, 0,  1,  // vertex number 0
            1, 0, -1,  // vertex number 1
           -1, 0, -1,  // vertex number 2
           -1, 0,  1,  // vertex number 3
            0, 1,  0   // vertex number 4
    ] );

pyramidFaceIndexArray = [
            3, 2, 1,  // First triangle in the base.
            3, 1, 0,  // Second Triangle in the base.
            3, 0, 4,  // Front face.
            0, 1, 4,  // Right face.
            1, 2, 4,  // Back face.
            2, 3, 4   // Left face.
    ];

pyramidGeom = new THREE.BufferGeometry();
pyramidGeom.setAttribute("position",
              new THREE.BufferAttribute(pyramidVert
pyramidGeom.setIndex( pyramidFaceIndexArray );
```

first vertex in group (index in vertex array if basic representation was used / 3)

number of vertices in group (number of faces in group x 3)

index in material array

```
pyramidGeom.addGroup(0,6,0);  // The base (2 triangles)
pyramidGeom.addGroup(6,3,1);  // Front face.
pyramidGeom.addGroup(9,3,2);  // Right face.
pyramidGeom.addGroup(12,3,3); // Back face.
pyramidGeom.addGroup(15,3,4); // Left face.
```

```
pyramidMaterialArray= [
          // Array of materials, for use as pyramids's material.
        new THREE.MeshLambertMaterial( { color: 0xffffff } ),
        new THREE.MeshLambertMaterial( { color: 0x99ffff } ),
        new THREE.MeshLambertMaterial( { color: 0xff99ff } ),
        new THREE.MeshLambertMaterial( { color: 0xffff99 } ),
        new THREE.MeshLambertMaterial( { color: 0xff9999 } )
      ];
```

```
pyramid = new THREE.Mesh( pyramidGeom, pyramidMaterialArray );
```


Modeling – Polymeshes

- different materials can be assigned to different faces
  - `BoxGeometry` comes with material indexes already assigned (in the order +x, -x, +y, -y, +z, -z)

```
const cubeMaterialArray =  [  // Array of materials, for use
          // Note: polygonOffset is used here to allow
        new THREE.MeshLambertMaterial( { color: "red"         } ),     // +x face
        new THREE.MeshLambertMaterial( { color: "cyan"        } ),     // -x face
        new THREE.MeshLambertMaterial( { color: "green"       } ),     // +y face
        new THREE.MeshLambertMaterial( { color: "magenta"     } ),     // -y face
        new THREE.MeshLambertMaterial( { color: "yellow"      } ),     // +z face
        new THREE.MeshLambertMaterial( { color: "blue"        } )      // -z face
    ];
```

```
let cubeGeom = new THREE.BoxGeometry(2,2,2);
cube = new THREE.Mesh( cubeGeom, cubeMaterialArray );
```


Modeling – Polymeshes

- different colors can be assigned to different vertices
  - set `color` attribute for geometry
  - set `vertexColors` property in the material to `true`

```
pyramidGeom.setAttribute(
        "color",
        new THREE.BufferAttribute( new Float32Array([
                1,1,1,  1,1,1,  1,1,1, // Base vertices are white
                1,1,1,  1,1,1,  1,1,1,
                1,0,0,  1,0,0,  1,0,0, // Front face vertices are red,
                0,1,0,  0,1,0,  0,1,0, // Right face vertices are green,
                0,0,1,  0,0,1,  0,0,1, // Back face vertices are blue,
                1,1,0,  1,1,0,  1,1,0  // Left face vertices are yellow.
             ]), 3)
      );
pyramid = new THREE.Mesh(
        pyramidGeom,
        new THREE.MeshLambertMaterial({
              color: "white",
              vertexColors: true
         })
      );
```

basic representation

vertices can have different colors in different faces, and the vertices within one face can have different colors

with vertex colors, the material color is taken as a multiplier for the specified vertex colors, so it is typically `white`

this works similarly for an indexed face set representation – the color buffer has one set of RGB values per vertex, so each vertex has the same color in every face it is part of

Supplying Texture Coordinates

```
let pyramidVertices = new Float32Array( [
            // Data for the pyramidGeom "position" attribute.
            // Contains the x,y,z coordinates for the vertices.
            // Each group of three numbers is a vertex;
            // each group of three vertices is one face.
          -1,0,1,  -1,0,-1,  1,0,-1, // First triangle in the base.
          -1,0,1,   1,0,-1,  1,0,1,  // Second triangle in the base.
          -1,0,1,   1,0,1,   0,1,0,  // Front face.
           1,0,1,   1,0,-1,  0,1,0,  // Right face.
           1,0,-1, -1,0,-1,  0,1,0,  // Back face.
          -1,0,-1, -1,0,1,   0,1,0   // Left face.
        ] );
let pyramidGeom = new THREE.BufferGeometry()
pyramidGeom.setAttribute("position",
                new THREE.BufferAttribute(py
```

```
let pyramidUVs = new Float32Array([
          0,0,  0,1,  1,1,   // uv coords for first triangle in base.
          0,0,  1,1,  1,0,   // uv coords for second triangle in base.
          0,0,  1,0,  0.5,1, // uv coords for front face.
          1,0,  0,0,  0.5,1, // uv coords for right face.
          0,0,  1,0,  0.5,1, // uv coords for back face.
          1,0,  0,0,  0.5,1  // uv coords for left face.
]);
pyramidGeom.setAttribute("uv",
                          new THREE.BufferAttribute(pyramidUVs,2) );
```

- texture coordinates are part of `BufferGeometry`
- attribute is `uv`
- value is an array of sets of texture coordinates
  - matches the `position` attribute for the basic mesh representation or the equivalent for an indexed face set

Loading Models

- mesh definitions can be loaded from a file
  - GLTF is the preferred format
    - extension `.gltf` or `.glb`
- `three.js` has utility functions for loading GLTF and other formats
  - not part of the `three.js` core

```
import { GLTFLoader } from "addons/loaders/GLTFLoader.js";
```

refers to the import map

```
<script type="importmap">
{
  "imports": {
    "three": "../threejs/three.module.min.js",
    "addons/": "../threejs/"
  }
}
</script>
```


Loading Models

```
loader = new GLTFLoader();
loader.load( url, onLoad, onProgress, onError );
```

starts the loading

- `url` – URL for the file to load (local relative path or `http://`)
- `onLoad` – callback function called when loading is complete
  - `load` is asynchronous and returns immediately

```
function onLoad(data) { // the parameter is the loaded model data
    let object = data.scene.children[0];
    // maybe modify the modeling transformation or material...
    scene.add(object);  // add the loaded object to our scene
    render();  // call render to show the scene with the new object
}
```

assumes the model file contains a `Mesh` object as the first child of the `Scene` object

  - `data` – for GLTF files, a three.js `Scene`
  - a `LoadingManager` is needed if the GLTF file involves external resources such as textures
    - `https://threejs.org/docs/#api/en/loaders/managers/LoadingManager`
- `onProgress` – callback called periodically during loading
  - value is `undefined` if no callback is desired
- `onError` – callback used if an error occurs