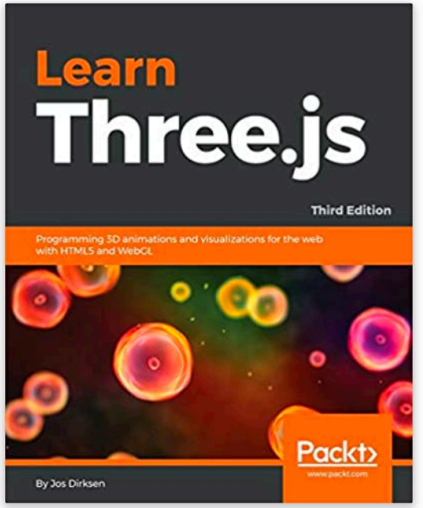
COMPUTER GRAPHICS



*Heavily based on CISC 3620 material by Prof. Michael Mandel

LAB – INTRODUCTION TO THREE.JS PROGRAMMING

Based on CS 307 lecture 2b

Topics for Today

- Exercises:
  - Changing width of barn
  - Two barns
  - A church
  - A hexagon (optional, to complete on your own)

Using Variables

- **Exercise: changing barn width**:
  - Start from this codepen
    - Fork the file
  - Edit the new file: change width of the barn (barnWidth)
    - Set new width to 40.
  - View the changed codepen
    - Verify your results

Warm-up Exercise: Changing Width

- Edit the pen again.
  - This time, mispell barnWidth somewhere, just to see what errors look like.
  - In your browser, open the JavaScript console
    - In Mac – go to 'Tools' – 'Web developer' – 'Web console'
  - Re-load the buggy pen and view the error message(s) in the JS console.
- Edit the pen to fix the spelling error.

Exercise: changing barn width

- Change the width of the barn again
  - This time, put numeric constants in place of the variable references.
  - Use a larger value for the width of the barn in the call to TW.createBarn()
    - versus the maxx used in the *bounding box* supplied in the call to TW.cameraSetup().

Warm-up Exercise: Changing Width

- How does the updated wide barn look?
- In this solution, the barn is wide (40) but maxx for the camera is only 20
  - => the camera setup is off.
- The original code allowed changing the width of the barn by changing one variable
  - => the camera setup changed automatically.
- => *variables may be more usable than numeric constants!*

Warm-up Exercise: Changing Width

- Change the numeric constants back to the variable barnWidth.
- The bounding box for the original barn truncates some of the roof of the barn from view.
  - Modify the value for maxy in the bounding box, so that the entire roof is visible.
    - Examine the createBarn() function: how is the y coordinate of the roof of the barn defined?

UPDATING THE BARN BUILDING

Updating the Barn Building

- Suppose we re-write the function that builds the barn
  - separating the creation of the list of vertices from the creation of the faces:
  - Updated barn

Updating the Barn Building

- function createBarnVertices(w, h, len) {
  - var barnGeometry = new THREE.Geometry();
  - // add the front
  - barnGeometry.vertices.push(new THREE.Vector3(0, 0, 0));
    barnGeometry.vertices.push(new THREE.Vector3(w, 0, 0));
    barnGeometry.vertices.push(new THREE.Vector3(w, h, 0));
    barnGeometry.vertices.push(new THREE.Vector3(0, h, 0));
    barnGeometry.vertices.push(new THREE.Vector3(0.5 * w, h + 0.5 * w, 0)); //
  - // just add the back also manually
  - barnGeometry.vertices.push(new THREE.Vector3(0, 0, -len));
    barnGeometry.vertices.push(new THREE.Vector3(w, 0, -len));
    barnGeometry.vertices.push(new THREE.Vector3(w, h, -len));
    barnGeometry.vertices.push(new THREE.Vector3(0, h, -len));
    barnGeometry.vertices.push(new THREE.Vector3(0.5 * w, h + 0.5 * w, -len)); //
  - return barnGeometry;
- }

Updating the Barn Building

- function createBarnFaces(barnGeometry) {

```
// now that we've got the vertices we need to define the faces.
// front faces
barnGeometry.faces.push(new THREE.Face3(0, 1, 2));
barnGeometry.faces.push(new THREE.Face3(0, 2, 3));
barnGeometry.faces.push(new THREE.Face3(3, 2, 4)); // // back faces
barnGeometry.faces.push(new THREE.Face3(5, 7, 6));
barnGeometry.faces.push(new THREE.Face3(5, 8, 7));
barnGeometry.faces.push(new THREE.Face3(7, 8, 9)); // // roof faces.
barnGeometry.faces.push(new THREE.Face3(3, 4, 8));
barnGeometry.faces.push(new THREE.Face3(4, 9, 8));
barnGeometry.faces.push(new THREE.Face3(2, 7, 9));
barnGeometry.faces.push(new THREE.Face3(4, 2, 9)); // // side faces
barnGeometry.faces.push(new THREE.Face3(6, 2, 1));
barnGeometry.faces.push(new THREE.Face3(7, 2, 6));
barnGeometry.faces.push(new THREE.Face3(0, 3, 5));
barnGeometry.faces.push(new THREE.Face3(3, 8, 5)); // // floor faces
barnGeometry.faces.push(new THREE.Face3(0, 5, 1));
barnGeometry.faces.push(new THREE.Face3(5, 6, 1)); // // calculate the normals for shading
barnGeometry.computeFaceNormals();
barnGeometry.computeVertexNormals(true); //
return barnGeometry;
```

- }

Updating the Barn Building

- This would give us the opportunity to modify the vertices before building the faces.
- Using this code:
  - var barnGeom = createBarnVertices(30,40,50);
    modifyVertices(barnGeom.vertices);
  - createBarnFaces(barnGeom);
  - ...
  - var barnMesh = TW.createMesh(barnGeom);
  - scene.add(barnMesh);

Updating the Barn Building

- How could we modify a vertex?
  - Using the THREE.Vector3 object properties ***x***, ***y***, and ***z***.
- In a JavaScript console in one of your windows, try the following
  - enter the code statements one at a time:
    - var p = new THREE.Vector3(1,2,3);
    - p.length();
    - p.x;
    - p.x = 10;
    - p.length()
  - What does the length() method appear to do?

Translation Exercise

- With a partner, examine the following JavaScript function - what does the function do?
  - function translateX(vertices,deltax) {
    - var len = vertices.length;
    - for(var i = 0; i < len ; i++) {
      - vertices[i].x += deltax;
    - }
  - }

TWO BARNS

Exercise: Two Barns

- Start from this codepen
- The file contains definition of translateX() function
  - creates a geometry and mesh for a single barn that is added to the scene
    - barn1geom and barn1mesh
- Modify the code to add a *second barn* that is:
  - *half the size* of the first barn
  - *shifted to the left* of the first barn, leaving a gap between the two barns

Exercise: Two Barns (cont.)

- Adjust the bounding box so that the two barns in are visible in their entirety
  - How does your result look like?

Object Origins

- We can avoid transforming all of the vertices individually
- Can use the Mesh.position property with a method set(x,y,z)
  - can be used to set its components
  - Property actually belongs to an instance of Object3D
    - the parent class of Mesh
- Thus, we can position our barn using the following code:
  - barn2mesh.position.set(-30,0,0);

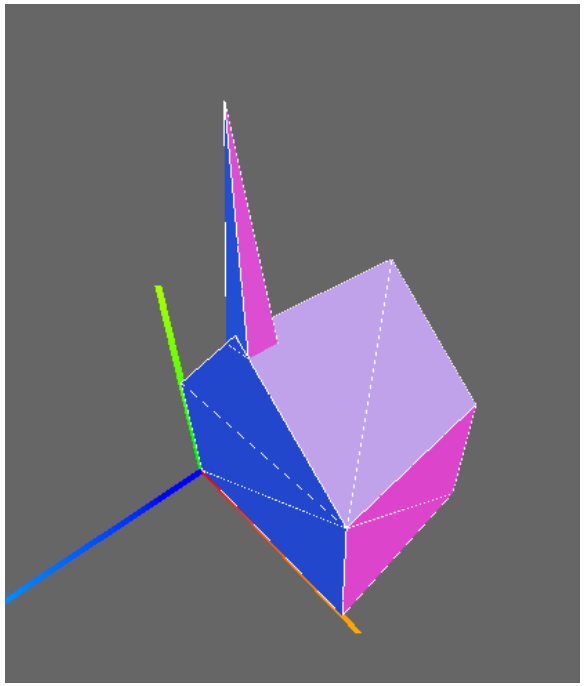
Object Origins

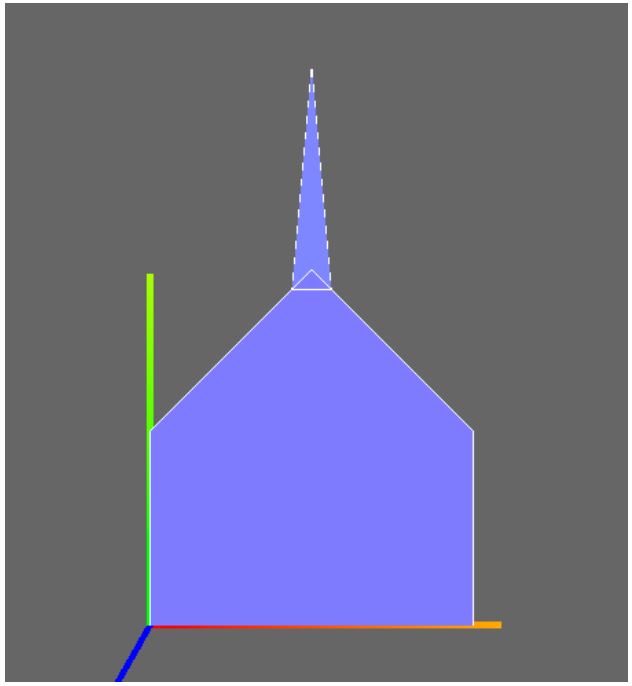
- This also avoids having to factor the createBarn() function the way we did
  - and having to create the translateX function
- How would the code look like?

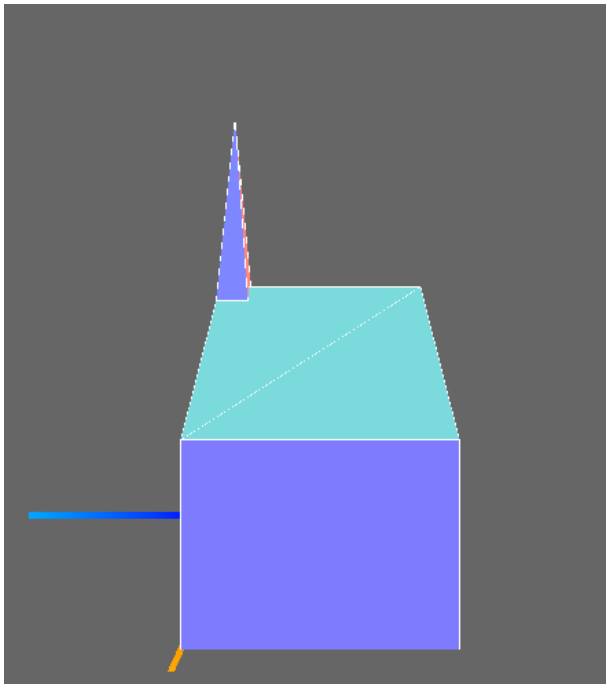
EXERCISES: CONVERT BARN TO CHURCH

Adding a Steeple

- Changes: adding a steeple to the barn to convert it into a church.
- The result will look like this:





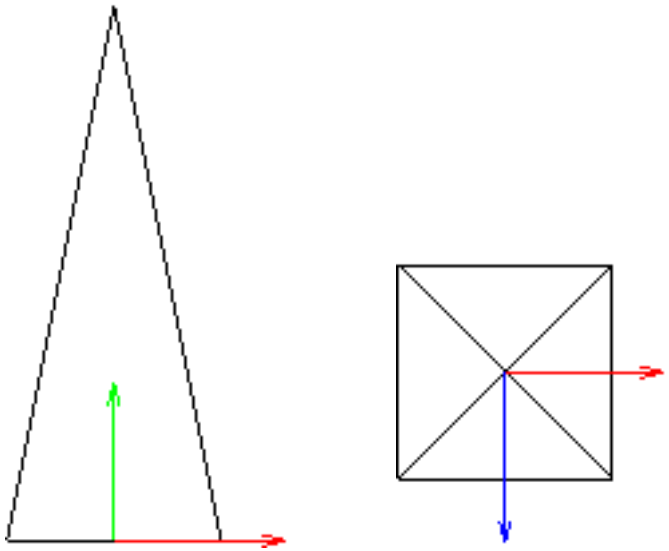


Adding a Steeple

- These pictures were based on the following:
  - The barn has width 50, height 30, and depth 40
  - The steeple is right in the middle of the ridge
  - The steeple's base is a square, 6 units on a side
  - The steeple is 36 units high, from base to tip

Adding a Steeple

- Here's now the steeple might look in "wireframe" from the front and from above:



Exercise: A Church, Part 1

- With a partner, figure out:
  - reasonable coordinates for a steeple of roughly the dimensions described above
    - Your coordinates need not be exact
    - Next week, we'll talk about how to use linear interpolation to get the coordinates exactly right
- Draft some code to draw the steeple.

Exercise: A Church, Part 2

- Start from this codepen
  - Create variables to store the dimensions of the steeple
  - Invoke the createSteeple() function
    - to create the geometry
  - Make a mesh using TW.createMesh()
  - Add the steeple to the scene
    - positioning it using position.set()
  - Adjust the bounding box in TW.cameraSetup()
    - so that you can see the entire church

A HEXAGON

A Hexagon (Optional)

- Barns in Pennsylvania Dutch country often have hex signs on them
  - They aren't hexagons, but ours will be.
- How would we draw a hexagon?
- More generally, how would we draw an N-gon?
- First, note that a hexagon is *flat* ,
  - so let's assume that we are drawing in the Z=k plane
  - our Z coordinate is always "k" and we only have to worry about X and Y.

A Hexagon (Optional)

- One way: iterate from 0 to the number of vertices. For each vertex:
- Compute the fraction of a full 2π circle that corresponds to this vertex.
  - The first (zeroth) vertex will always be at 0 degrees.
  - The second vertex of a hexagon will be 1/6th of the way around the circle, so $(\frac{1}{6}) * (2\pi)$
    - and so on.

A Hexagon (Optional)

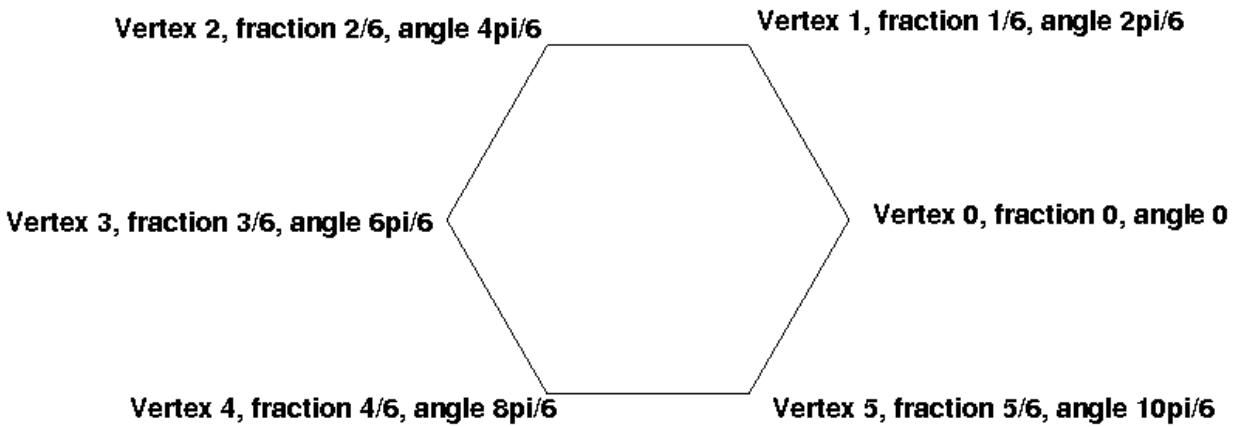
- Compute the angle for the vertex.
  - The second vertex of a hexagon will be at $\frac{2\pi}{6}$
    - or, $\frac{360}{6} = 60$ degrees.

A Hexagon (Optional)

- Compute the X and Y coordinates corresponding to that angle
  - We'll need the radius, too, to do this.
- This has the coordinates centered around the origin.
- Create a THREE.Vector3() object to record those coordinates
  - Do this for two different values for Z, and you can create a hexagonal cylinder!

A Hexagon (Optional)

- Here's an example:



- We will learn how to use *extrusions* to turn a 2D thing like this hexagon into a hexagonal cylinder.

Exercise: A Hexagon

- Working with a partner, draft some JavaScript to do the computations to make a hexagon
  - or more-generally, a regular polygon.

Exercise: A Hexagon

- Here is the start of such a function
- // Creates a regular polygon in a plane of constant z
- function createRegularPolygon(numVertices, radius, zCoordinate) {
  - var geom = new THREE.Geometry();
  - var i;
  - for(i = 0; i < numVertices; ++i) {
    - var fraction = i / numVertices;
    - var angle = 2 * Math.PI * fraction; // in radians
    - var x = radius * Math.cos(angle);
    - var y = radius * Math.sin(angle);
    - geom.vertices.push(new THREE.Vector3(x,y,zCoordinate));
  - } // compute faces ...
  - return geom;
- }

Coding Advice

- Build and test your code *incrementally*. Save often!
- Save versions by forking your codepen
  - Or by downloading versions periodically
  - Or by downloading and using version control!
- It will be easier to experiment with things if you know you can go back to an earlier version.

Coding Advice

- Build and test your code *incrementally*. Save often!
- Save versions by forking your codepen
  - Or by downloading versions periodically
  - Or by downloading and using version control!
- It will be easier to experiment with things if you know you can go back to an earlier version.

Coding Advice

- Be willing to create a simple “test” program
  - to see how something works without all the complexity of your larger program.
- Be modular, and document as you go. It’ll be easier to understand and debug your own code.

Questions?