EDAF80 Introduction to Computer Graphics

Seminar 2

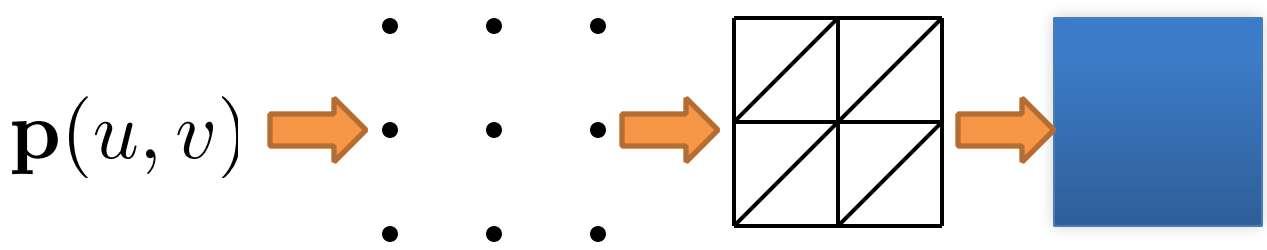
Tessellation & Interpolation

Pierre Moreau

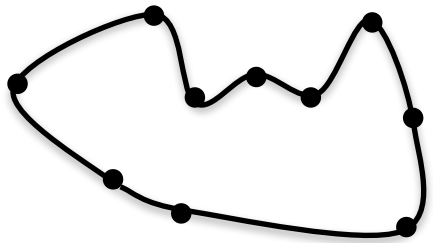
2019 Includes slides by Carl Johan Gribel, 2010-13 and Michael Dogget, 2014-17

Today

- Tessellation

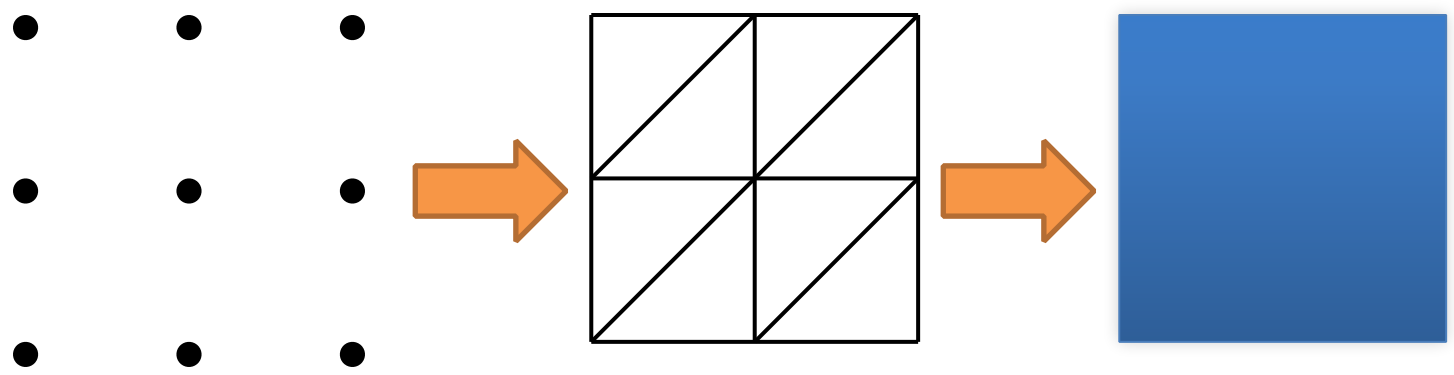


- Linear and cubic interpolation



- Assignment 2
- Git

Tessellation Walkthrough: Quad



1. Setup *vertex array*
2. Setup *index array* (triangulate)

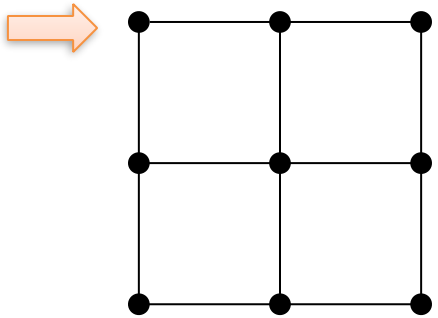
Quad: vertex array creation

- **Create vertex array** (e.g. 3x3 vertices)

```
auto vertices  = std::vector<glm::vec3>(9);
```

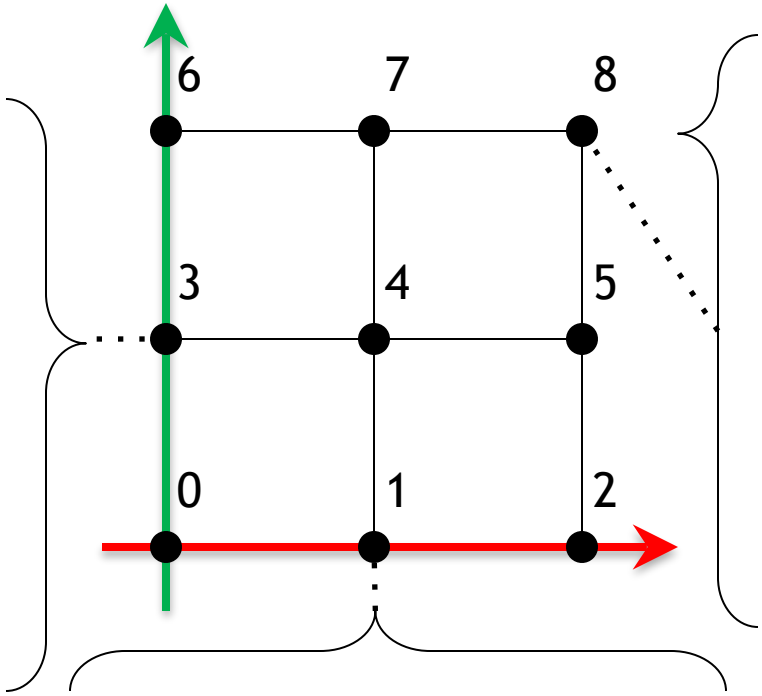
- **Assign one vertex**

```
vertices[index] = glm::vec3(x,y,z);
```



Quad: vertex array setup

- **Set vertices**




```
vertices[3].x = 0
vertices[3].y = 0.5
vertices[3].z = 0
```

```
vertices[8].x = 1
vertices[8].y = 1
vertices[8].z = 0
```

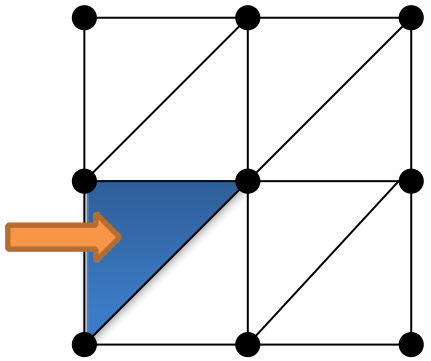
```
vertices[1].x = 0.5
vertices[1].y = 0
vertices[1].z = 0
```

Quad: Triangulation

- **Create index array** (2(3-1)(3-1) triangles)

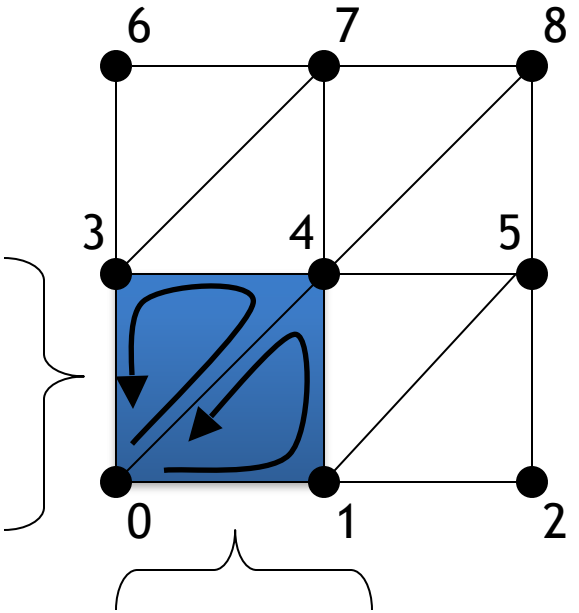
```
auto indices = std::vector<glm::uvec3>(8);
```

- **Define triangle** (indices for three vertices)
  - `indices[index] = glm::vec3(x,y,z);`



Quad: Triangulation

- Indices in counter-clockwise order (*ccw*)

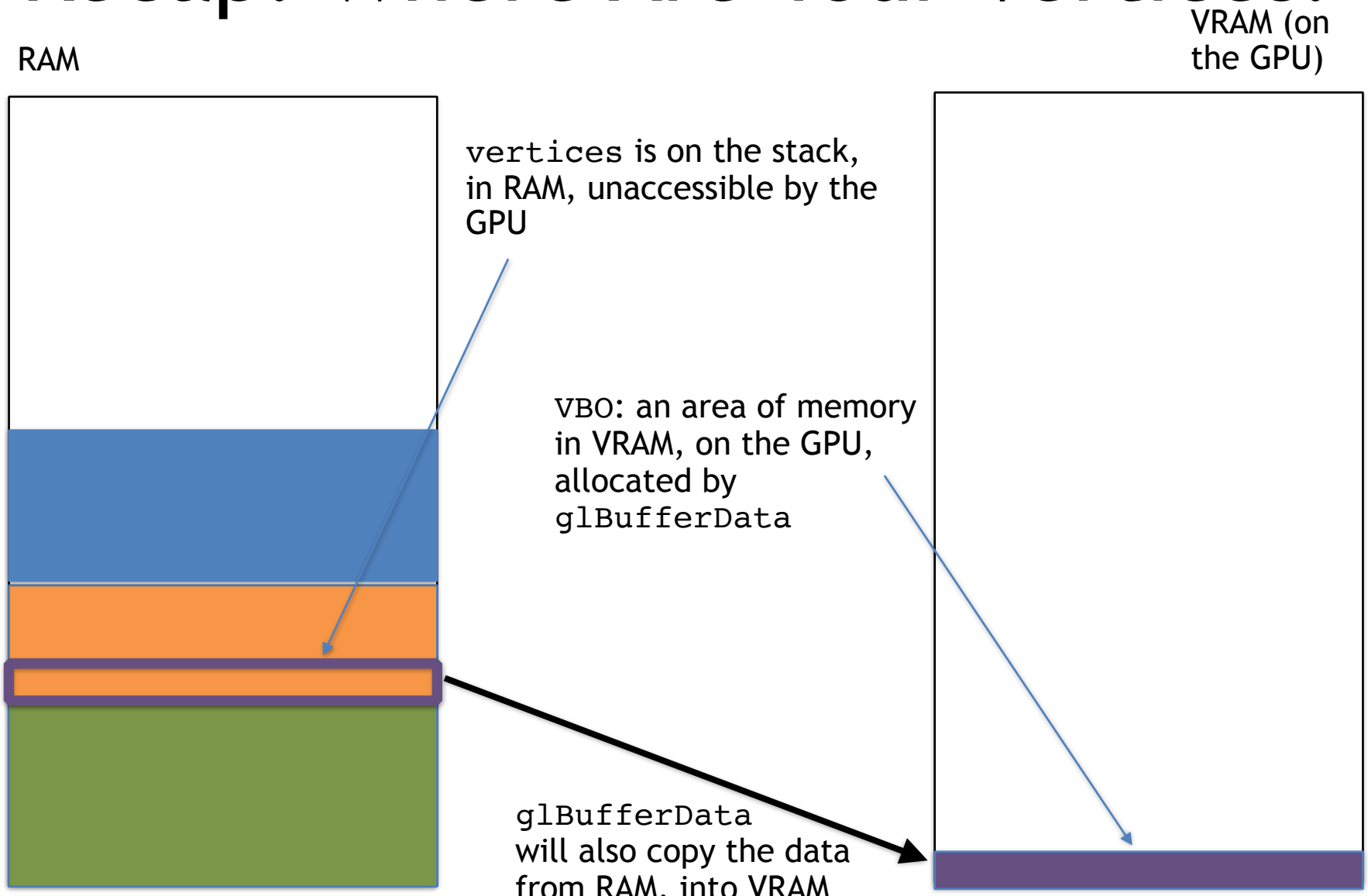



```
indices[1].x = 0;
indices[1].y = 4;
indices[1].z = 3;
```

```
indices[0].x = 0;
indices[0].y = 1;
indices[0].z = 4;
```

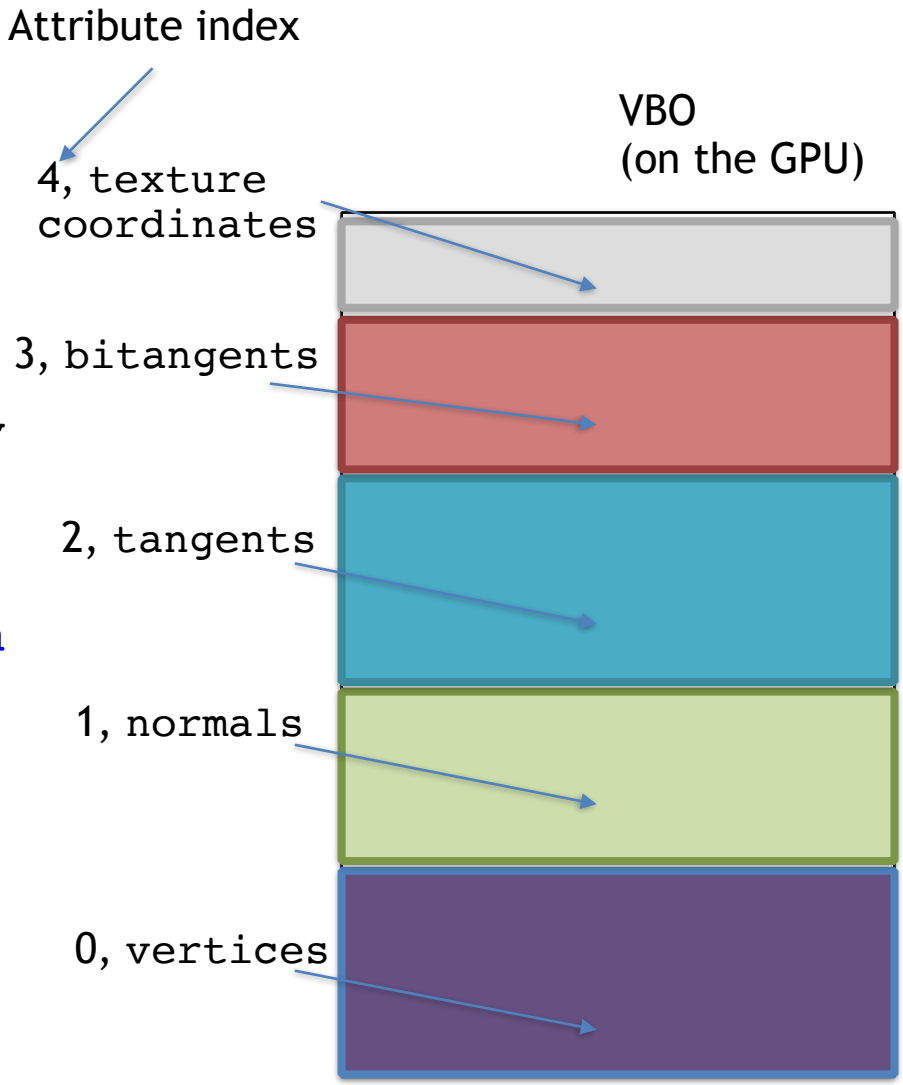
Backface culling is off by default
Turn it on to improve performance!

Recap: Where Are Your Vertices?

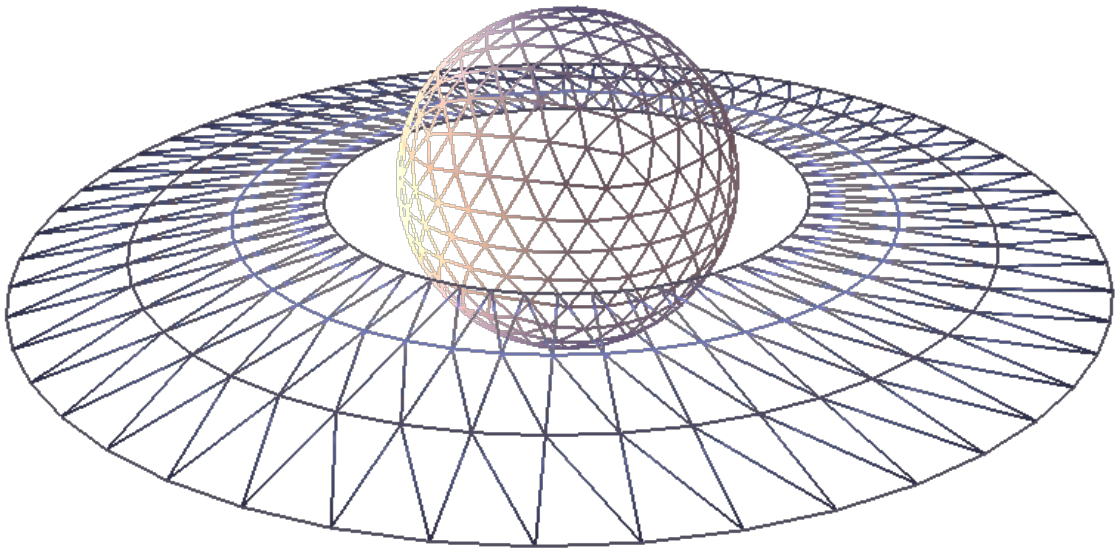


Placing the vertex attributes

- Same VBO used for multiple attributes
- Attributes stored in VAO (Vertex Array Object)
- Enabled using `glEnableVertexAttribArray(uint attribute_index)`
- Configured via `glVertexAttribPointer(uint attribute_index, uint component_count, GLenum component_type, GLbool normalize, GLsize stride, GLvoid const* offset)`

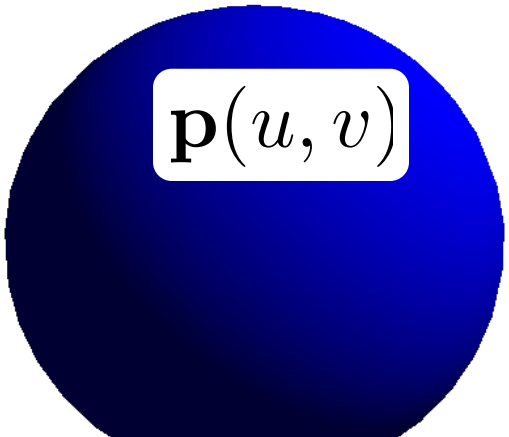


Parametric surfaces



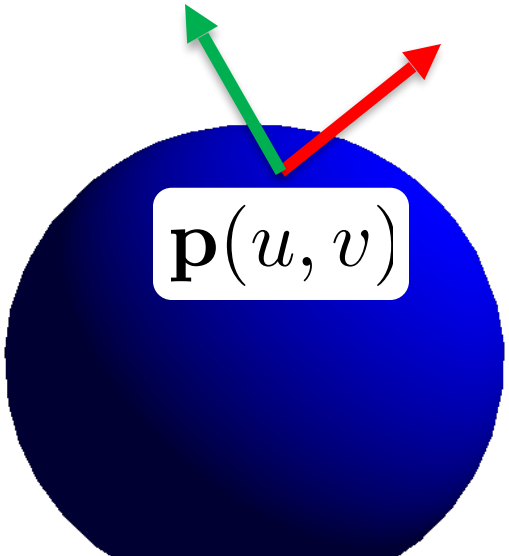
Parametric surface & tangent space

- Map surface from 2D: $\mathbf{p}(x, y, z) = \mathbf{p}(u, v)$
$: \mathbb{R}^2 \mapsto \mathbb{R}^3$



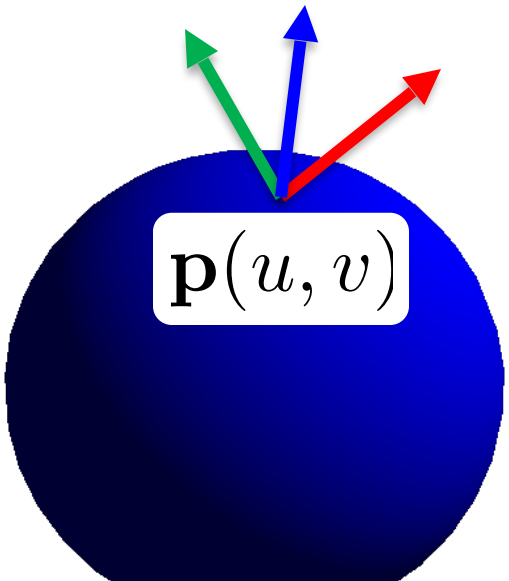
Parametric surface & tangent space

- Map surface from 2D: $\mathbf{p}(x, y, z) = \mathbf{p}(u, v)$
  $: \mathbb{R}^2 \mapsto \mathbb{R}^3$
  - tangent $\mathbf{t} = \dfrac{\partial \mathbf{p}}{\partial u}$
  - binormal $\mathbf{b} = \dfrac{\partial \mathbf{p}}{\partial v}$



Parametric surface & tangent space

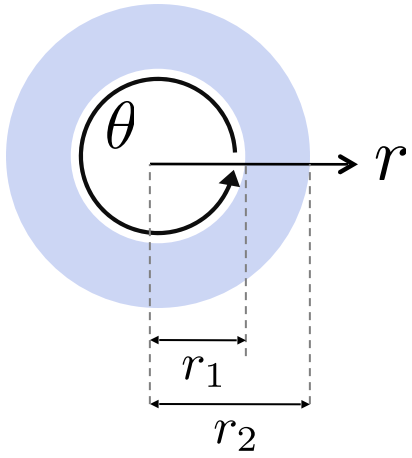
- Map surface from 2D: $\mathbf{p}(x, y, z) = \mathbf{p}(u, v)$
  $: \mathbb{R}^2 \mapsto \mathbb{R}^3$
  - tangent $\mathbf{t} = \dfrac{\partial \mathbf{p}}{\partial u}$
  - binormal $\mathbf{b} = \dfrac{\partial \mathbf{p}}{\partial v}$
  - normal $\mathbf{n} = \dfrac{\partial \mathbf{p}}{\partial u} \times \dfrac{\partial \mathbf{p}}{\partial v}$

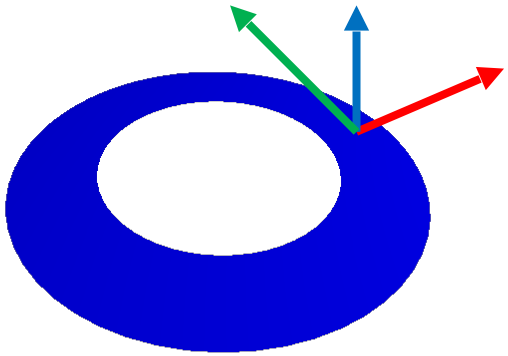


Circle ring

Surface $\mathbf{p}(r,\theta) = \begin{Bmatrix} r\cos(\theta) \\ r\sin(\theta) \\ 0 \end{Bmatrix}, r_1 \le r \le r_2, 0 \le \theta < 2\pi$

$\mathbf{t} = \dfrac{\partial \mathbf{p}}{\partial r} = \begin{Bmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \end{Bmatrix}$ $\mathbf{b} = \dfrac{\partial \mathbf{p}}{\partial \theta} = \begin{Bmatrix} -r\sin(\theta) \\ r\cos(\theta) \\ 0 \end{Bmatrix}$

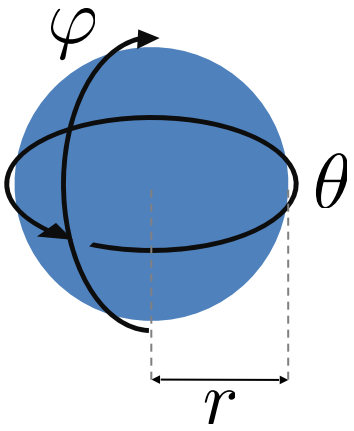


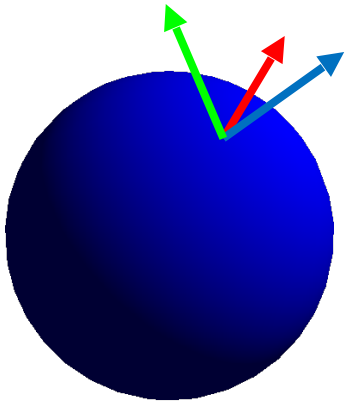


Sphere

$$\text{Surface } \mathbf{p}(\theta,\varphi) = \begin{Bmatrix} r\sin(\theta)\sin(\varphi) \\ -r\cos(\varphi) \\ r\cos(\theta)\sin(\varphi) \end{Bmatrix}, 0 \le \theta \le 2\pi, 0 \le \varphi \le \pi$$

$$\mathbf{t} = \frac{\partial \mathbf{p}}{\partial \theta} = \begin{Bmatrix} r\cos(\theta)\sin(\varphi) \\ 0 \\ -r\sin(\theta)\sin(\varphi) \end{Bmatrix} \quad \mathbf{b} = \frac{\partial \mathbf{p}}{\partial \varphi} = \begin{Bmatrix} r\sin(\theta)\cos(\varphi) \\ r\sin(\varphi) \\ r\cos(\theta)\cos(\varphi) \end{Bmatrix}$$





Sphere

Surface $\mathbf{p}(\theta,\varphi) = \begin{Bmatrix} r\sin(\theta)\sin(\varphi) \\ -r\cos(\varphi) \\ r\cos(\theta)\sin(\varphi) \end{Bmatrix}, 0 \le \theta \le 2\pi, 0 \le \varphi \le \pi$

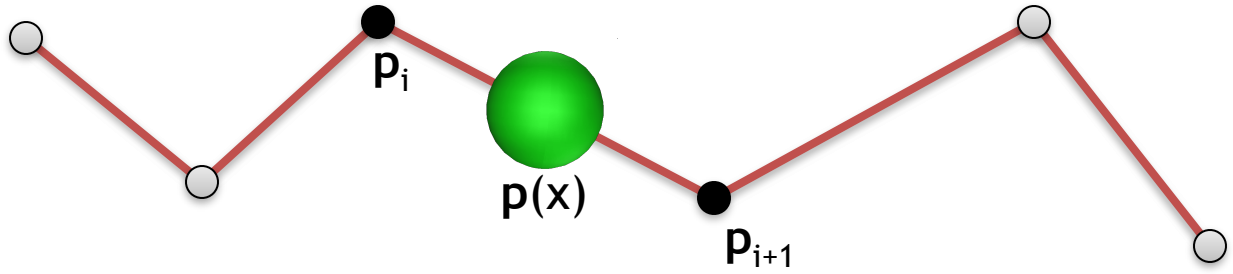
$\mathbf{t} = \dfrac{\partial \mathbf{p}}{\partial \theta} = \begin{Bmatrix} r\cos(\theta)\sin(\varphi) \\ 0 \\ -r\sin(\theta)\sin(\varphi) \end{Bmatrix}$ $\mathbf{b} = \dfrac{\partial \mathbf{p}}{\partial \varphi} = \begin{Bmatrix} r\sin(\theta)\cos(\varphi) \\ r\sin(\varphi) \\ r\cos(\theta)\cos(\varphi) \end{Bmatrix}$

- **t** and **b** can be simplified since only direction is important
- **t** *needs* to be simplified, or it will be undefined for ϕ=0

Today

- Tessellation ✔
- Linear and cubic interpolation
- Assignment 2

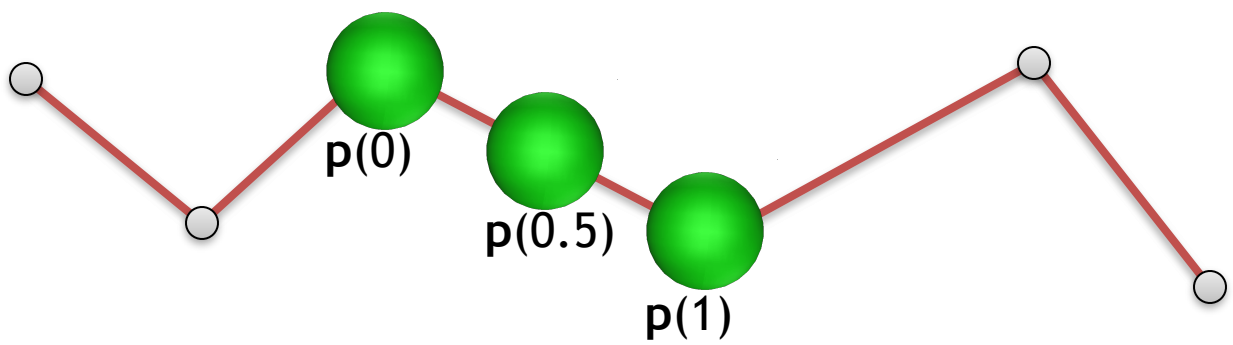
Linear interpolation (LERP)



$$\mathbf{p}(x) = \begin{bmatrix} 1 & x \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p_i} \\ \mathbf{p_{i+1}} \end{bmatrix}, x \in [0,1]$$

Linear interpolation (LERP)

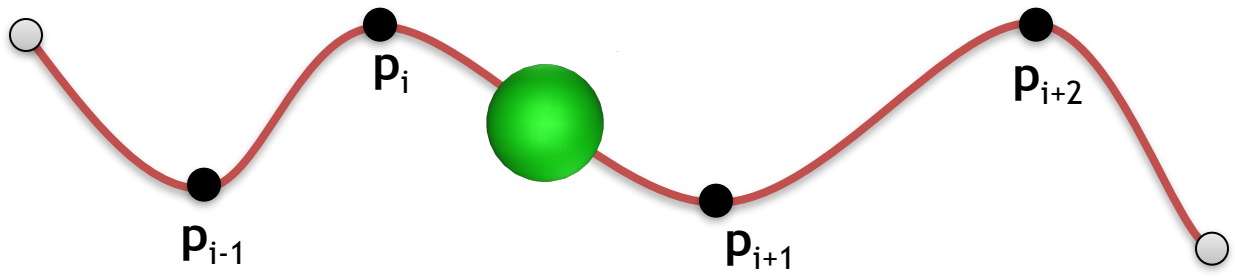
- Interpolate between $\mathbf{p_i}$ and $\mathbf{p_{i+1}}$ for **x = [0, 1]**



$$\mathbf{p}(x) = \begin{bmatrix} 1 & x \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p_i} \\ \mathbf{p_{i+1}} \end{bmatrix}, x \in [0, 1]$$

Cubic interpolation

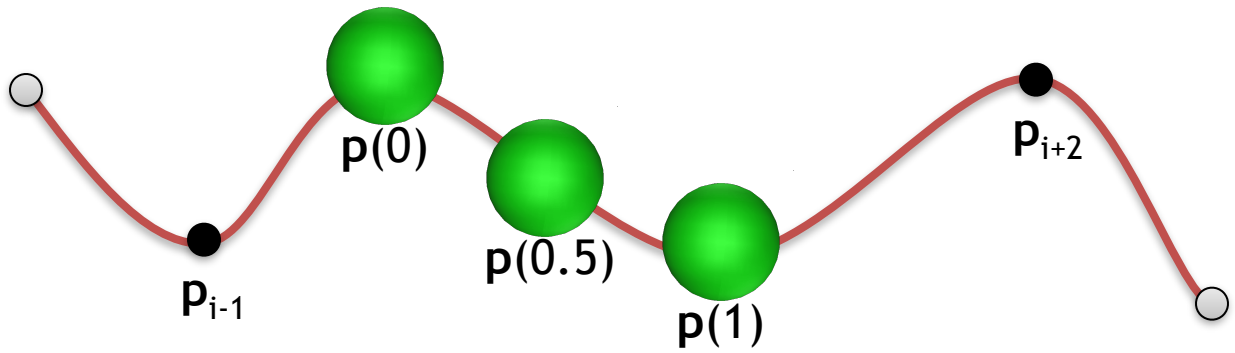
- Catmull-Rom spline



$$\mathbf{q}(x) = \begin{bmatrix} 1 & x & x^2 & x^3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau - 3 & 3 - 2\tau & -\tau \\ -\tau & 2 - \tau & \tau - 2 & \tau \end{bmatrix} \begin{bmatrix} \mathbf{p_{i-1}} \\ \mathbf{p_i} \\ \mathbf{p_{i+1}} \\ \mathbf{p_{i+2}} \end{bmatrix}, x \in [0, 1]$$

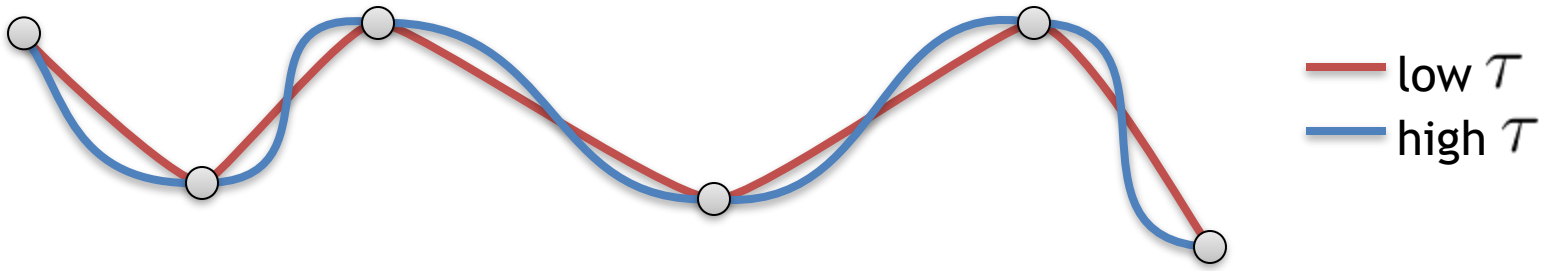
Cubic interpolation

- Interpolate between $\mathbf{p_i}$ and $\mathbf{p_{i+1}}$ for $\mathbf{x = [0, 1]}$, using $\mathbf{p_{i-1}}$, $\mathbf{p_i}$, $\mathbf{p_{i+1}}$, $\mathbf{p_{i+2}}$



$$\mathbf{q}(x) = \begin{bmatrix} 1 & x & x^2 & x^3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau - 3 & 3 - 2\tau & -\tau \\ -\tau & 2 - \tau & \tau - 2 & \tau \end{bmatrix} \begin{bmatrix} \mathbf{p_{i-1}} \\ \mathbf{p_i} \\ \mathbf{p_{i+1}} \\ \mathbf{p_{i+2}} \end{bmatrix}, x \in [0, 1]$$

Tension



$$\mathbf{q}(x) = \begin{bmatrix} 1 & x & x^2 & x^3 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau - 3 & 3 - 2\tau & -\tau \\ -\tau & 2 - \tau & \tau - 2 & \tau \end{bmatrix} \begin{bmatrix} \mathbf{p_{i-1}} \\ \mathbf{p_i} \\ \mathbf{p_{i+1}} \\ \mathbf{p_{i+2}} \end{bmatrix}, x \in [0, 1]$$

τ = tension, how sharply the curve bends at control points
Keep within [0,1]
Good initial value: **0.5**

Today

- Tessellation ✔
- Linear and cubic interpolation ✔
- Assignment 2

Take a break :)

- Unity Real-Time Adam demo
  - https://youtu.be/GXI0l3yqBrA

Assignment 2

1. Tessellate objects from parametric equation
2. Linear and cubic interpolation

Assignment 2: Tessellation

- Implement function bodies in **`src/EDAF80/parametric_shapes.cpp`** :

```
parametric_shapes::createQuad(…);
parametric_shapes::createSphere(…);
parametric_shapes::createTorus(…); (Optional)
```

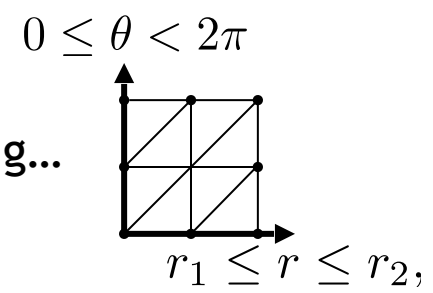
Assignment 2: Tessellation

- Implement function bodies in **`src/EDAF80/parametric_shapes.cpp`** :

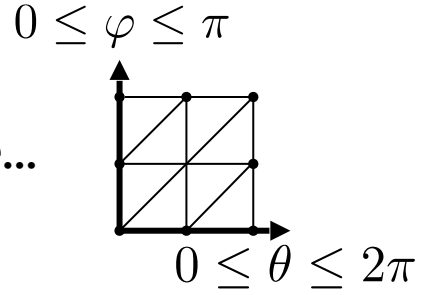
```
parametric_shapes::createSphere(...);
parametric_shapes::createSphere(…);
parametric_shapes::createTorus(…); (Optional)
```

- **Hint**
  Examine, in the same file: `createCircleRing(...);`
  Note: check parameter definitions & ranges

circle ring…



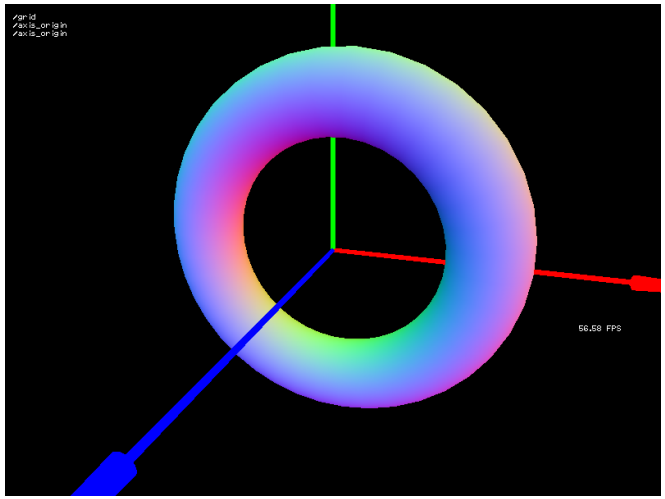
sphere…



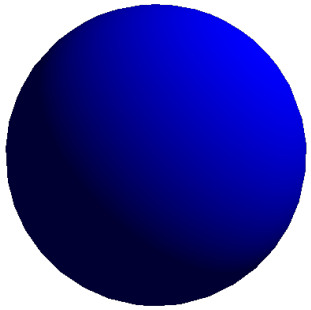
Assignment 2: Tessellation

Debugging normals/vectors

- Colorize
  - Use the ”*Normals*”-shader to represent normals as RGB-values



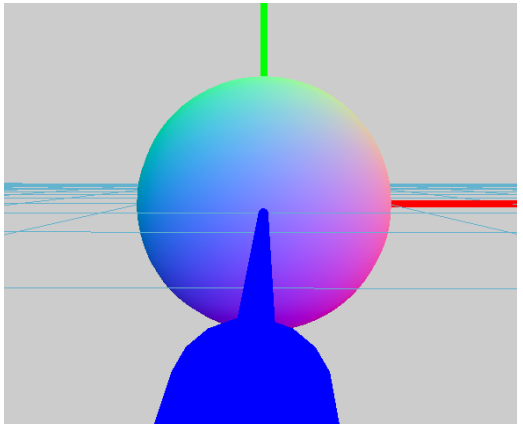
- Inspect illumination
  - Is illumination consistent with the location of the light source?



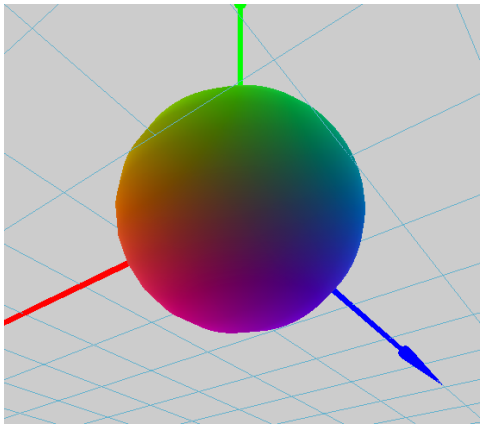
Normals shader on a sphere

- Map from [-1,1] to [0,1]
  - (normal * 0.5) + 0.5
- X axis (1,0,0) becomes (0.5,0.5,1)
- Y axis (-1,0,-1) is 0, 0.5, 0
- halfway between X&Z is (1,0.5,1)

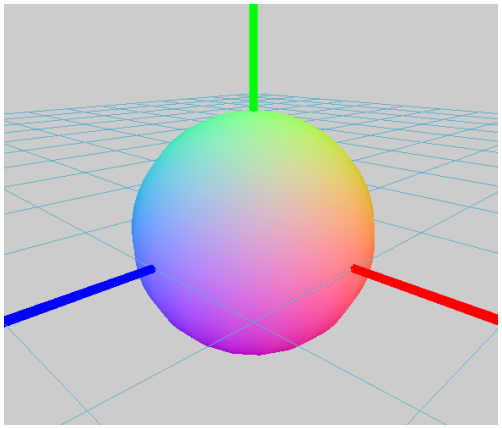
+X axis



(0.5,0.5,1)

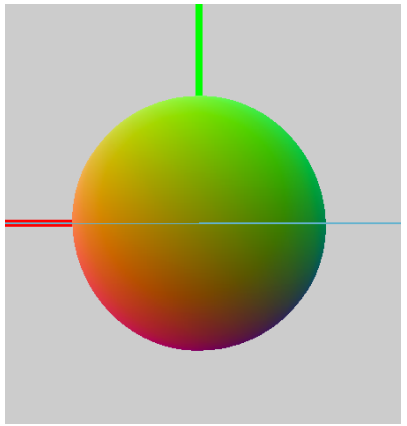


(0,0,0)



(1,1,1)

-X axis

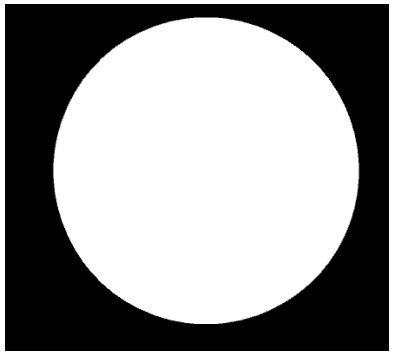


(0.5,0.5,0)

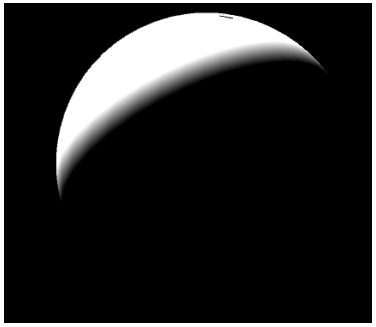
Scene Controls

- Shaders
  - Accessible from a dropdown in the GUI

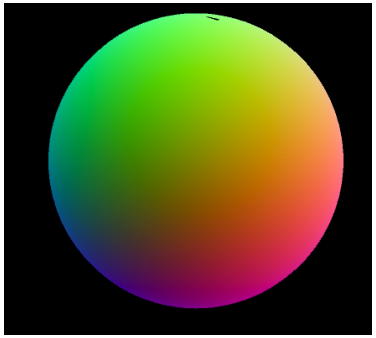
Default



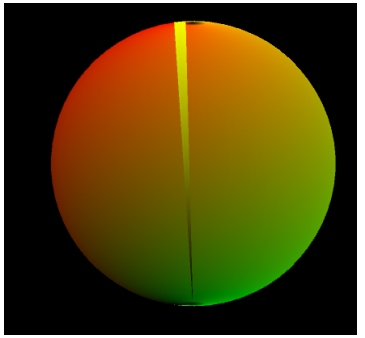
Diffuse



Normals



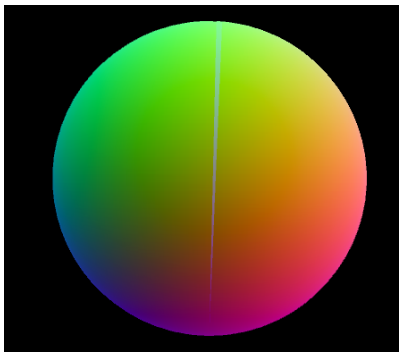
Texture Coords



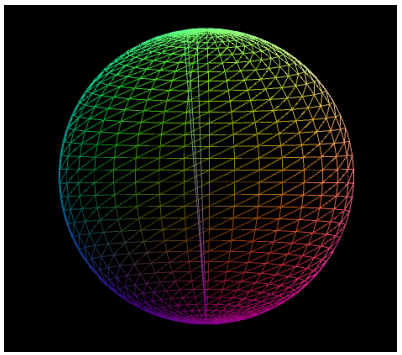
Scene Controls

- Polygon mode
  - Accessible from a dropdown in the GUI

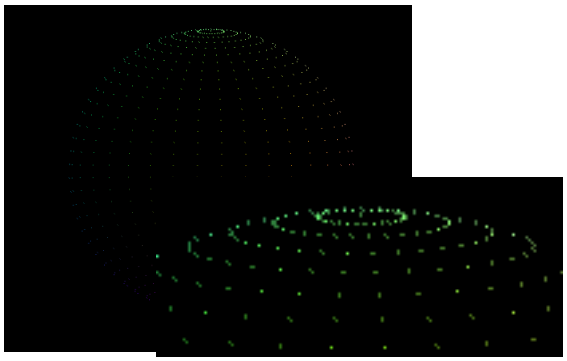
Filled



Line

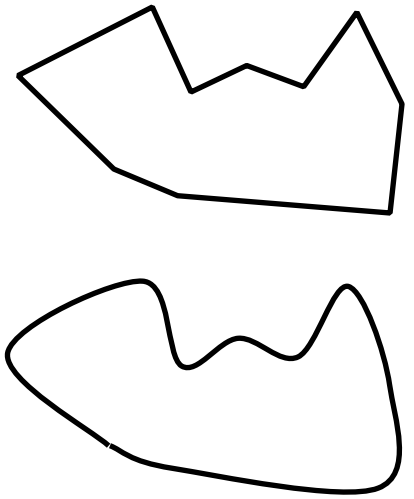


Point



Assignment 2

2. Interpolation



- Implement linear and cubic interpolation

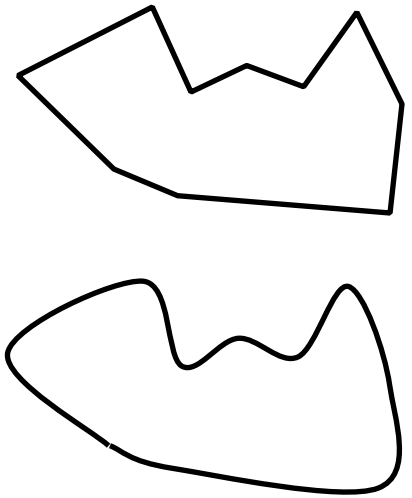
  Function bodies in
  `src/EDAF80/interpolation.cpp`:

```
glm::vec3 evalLERP(...);
glm::vec3 evalCatmullRom(...);
```

- Test with just **2** (lerp) or **4** (cubic) points first
- Then set up **cyclic path of >=10 points**;
  animate an object along the path using both functions
- Use **`catmull_rom_tension`** and **`use_linear`** variable
  - Then control them in GUI

Assignment 2

2. Interpolation



- Implementation sketch (cubic):

```
// Init:
glm::vec3 cp[N]; // N control points
float path_pos = 0.0f;
float pos_velocity = ...
// Main loop:
int i = floor(path_pos); // floor returns closest lower integer
// use indices (e.g.) i, i+1, i+2, i+3 to retrieve control
//     points from cp[].
// make sure indices wrap: 0, 1 … N-1, 0, 1…
// run interpolation
// update the animated object
path_pos += pos_velocity; // step forward
```

Today

- Tessellation ✔
- Linear and cubic interpolation ✔
- Assignment 2 ✔

OpenGL - More information

- learnopengl.com
- In particular “Hello Triangle”
  - http://www.learnopengl.com/#!Getting-started/Hello-Triangle
- Can also use **Elgkarv** computer lab
- Ask questions on Forum