More Drawing

Motivation

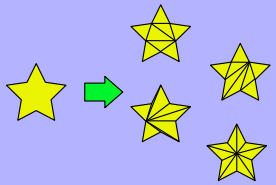
- We need to be able to render more than just points
- Today, we'll look at how to do so

GL

- GL can only draw four kinds of things:
  - Points
  - Lines
  - Triangles
  - Patches (we ignore these for now)
- Anything we want to draw must be made up of one of these
  - In practice, most of our drawing is of triangles

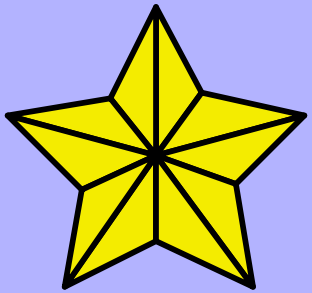
Tessellation

- Tessellation: To break up a figure into triangles
  - Any polygon can be expressed as one or more triangles
- Usually several ways to express any figure
  - Do we want equilateral triangles?
  - Can we add additional vertices?



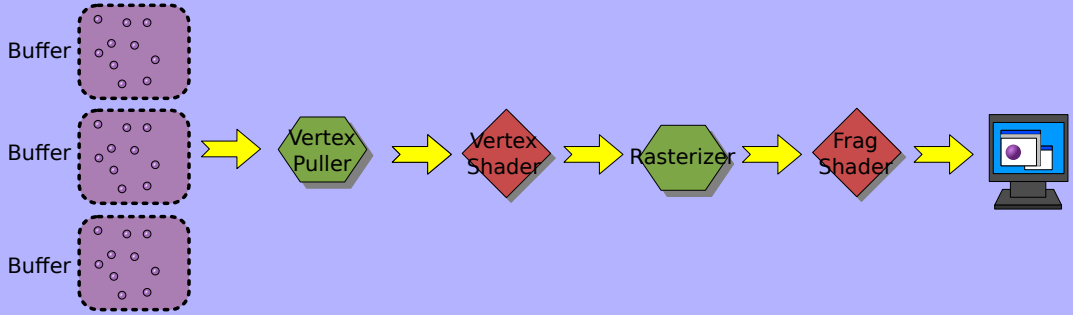
Notice

- In a tessellation, we often have some vertices shared by several triangles



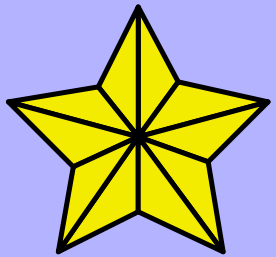
Pipeline

- Recall GPU pipeline:



VS

- Vertex shader runs three times per triangle
- So here, VS runs 30 times
- But there's only 6 unique vertices
- This represents wasted computation!

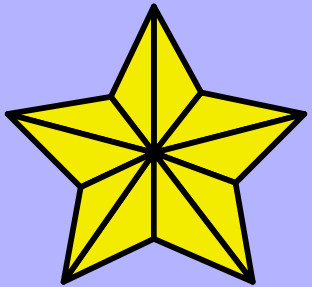


Solution

- GPU's support *indexed meshes*
- Now we have two buffers for the mesh
  - First buffer gives vertex positions
  - Second buffer tells how they are connected

Example

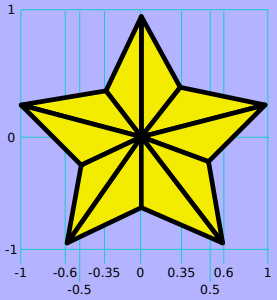
- Suppose we have this tessellated figure:



Vertex Buffer

- Here's the vertex buffer:

```
0.00,0.00
0.35,0.40
0.00,1.00
1.00,0.20
0.50,-0.25
-0.35,0.40
0.60,-0.95
-0.60,-0.95
0.00,-0.65
-1.00,0.20
-0.50,-0.25
```



- Trace out (in class) which points correspond to which VB entries

Index Buffer

- Here's the index buffer:

```
0 1 2
3 0 1
3 4 0
0 5 2
0 4 6
7 8 0
0 8 6
0 9 5
0 9 10
0 10 7
0 9 10
```

- Show (in class) how indices correspond to triangles

Code

- When we want to define an indexed mesh, we have a couple of extra steps
- First, define vertex data
  - This is just as we have done
  - `vbuff = Buffer( array.array(“f”,…data…) )`
- Now to create index buffer
  - `ibuff = Buffer( array.array(“I”,…data…) )`

Drawing

- We must tweak the VAO creation:

```
tmp = array.array("I",[0])
glGenVertexArrays(1,tmp)
vao = tmp[0]
glBindVertexArray(vao)
ibuff.bind(GL_ELEMENT_ARRAY_BUFFER)   # New!
vbuff.bind(GL_ARRAY_BUFFER)
glEnableVertexAttribArray(0)
glVertexAttribPointer( 0, 2, GL_FLOAT, False, 2*4, 0 )
glBindVertexArray(0)
```

- The VAO remembers which index buffer is bound, so that's also part of VAO state

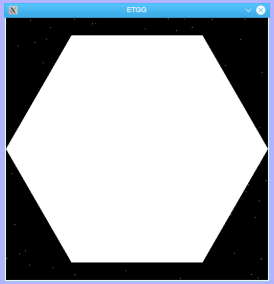
Rendering

- To render:
  glBindVertexArray(vao)
  glDrawElements(GL_TRIANGLES, count, GL_UNSIGNED_INT,0)
- First parameter = type to draw
- Second = how many vertices
- Third = what type of indices do we have
- Must match type used when buffer was created
- Fourth = byte offset in index buffer where data starts

Result

Assignment

- Display a hexagon on top of your starfield. The hexagon should be specified with indexed rendering
- Bonus [+25%]: Display a circle instead of a hexagon
- Your code must be efficient: Don't re-create buffers every frame, and don't leak memory!



Sources

- Various authors. Vertex Specification. https://www.khronos.org/opengl/wiki/Vertex_Specification#Vertex_Array_Object

Created using LaTeX.

Main font: Gentium Book Basic, by Victor Gaultney. See http://software.sil.org/gentium/
Monospace font: Source Code Pro, by Paul D. Hunt. See https://fonts.google.com/specimen/Source+Code+Pro and http://sourceforge.net/adobe
Icons by Ulisse Perusin, Steven Garrity, Lapo Calamandrei, Ryan Collier, Rodney Dawes, Andreas Nilsson, Tuomas Kuosmanen, Garrett LeSage, and Jakub Steiner. See http://tango-project.org