CSE 167

Discussion 04 ft. Joanna

10/24/2018

Announcements

- Project 3 is due 11/2 2PM
- Midterm I Thursday
    - Closed book / no cheat sheets

Contents

- Texture
  - Buffer and shader
- Scene graph
  - Class hierarchy
  - Class implementation
  - Example
- Midterm Review

Texture: buffer and shader

```
GLuint textureID;
glGenTextures(1, &textureID);

glBindTexture(GL_TEXTURE_2D, textureID);
unsigned char * image = loadPPM("myTexture.ppm", width, height);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, image);

glBindTexture(GL_TEXTURE_2D, 0);
```

Texture: buffer and shader

```
GLuint textureID;
glGenTextures(1, &textureID);

glBindTexture(GL_TEXTURE_2D, textureID);
unsigned char * image = loadPPM("myTexture.ppm", width, height);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, image);

glBindTexture(GL_TEXTURE_2D, 0);
```

- Generates the texture
  - Similar to “glGenBuffers”
- Let the GPU know that we are going to send a texture
  - “textureID” is the identifier of such texture

Texture: buffer and shader

```
GLuint textureID;
glGenTextures(1, &textureID);

glBindTexture(GL_TEXTURE_2D, textureID);
unsigned char * image = loadPPM("myTexture.ppm", width, height);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, image);

glBindTexture(GL_TEXTURE_2D, 0);
```

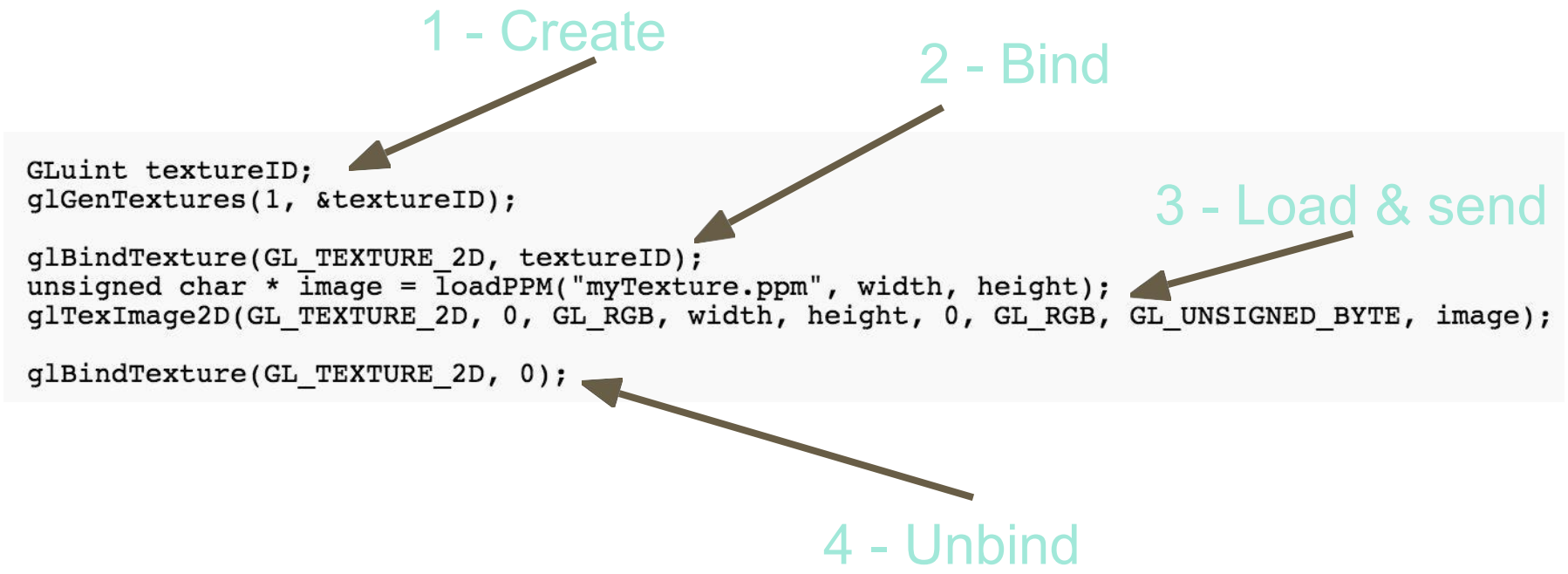
- Similar to VBOs, we need to bind the texture so OpenGL knows which texture we are modifying
  - (Reminder) OpenGL is a state machine!
- Highly recommended to unbind (bind texture 0) once you're done modifying the texture to avoid unexpected results

Texture: buffer and shader

```
GLuint textureID;
glGenTextures(1, &textureID);

glBindTexture(GL_TEXTURE_2D, textureID);
unsigned char * image = loadPPM("myTexture.ppm", width, height);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, image);

glBindTexture(GL_TEXTURE_2D, 0);
```

- Load texture and send it to the GPU
  - GL_TEXTURE_2D: type of texture
  - 0: mimap level
  - GL_RGB: Internal representation of the texture in the GPU
  - width/height: variable for the texture width/height
  - 0: should always be 0
  - GL_RGB: Representation of the texture that we are sending
  - GL_UNSIGNED_BYTE: Type of individual values in our image array
  - image: memory location where the pixel information is stored

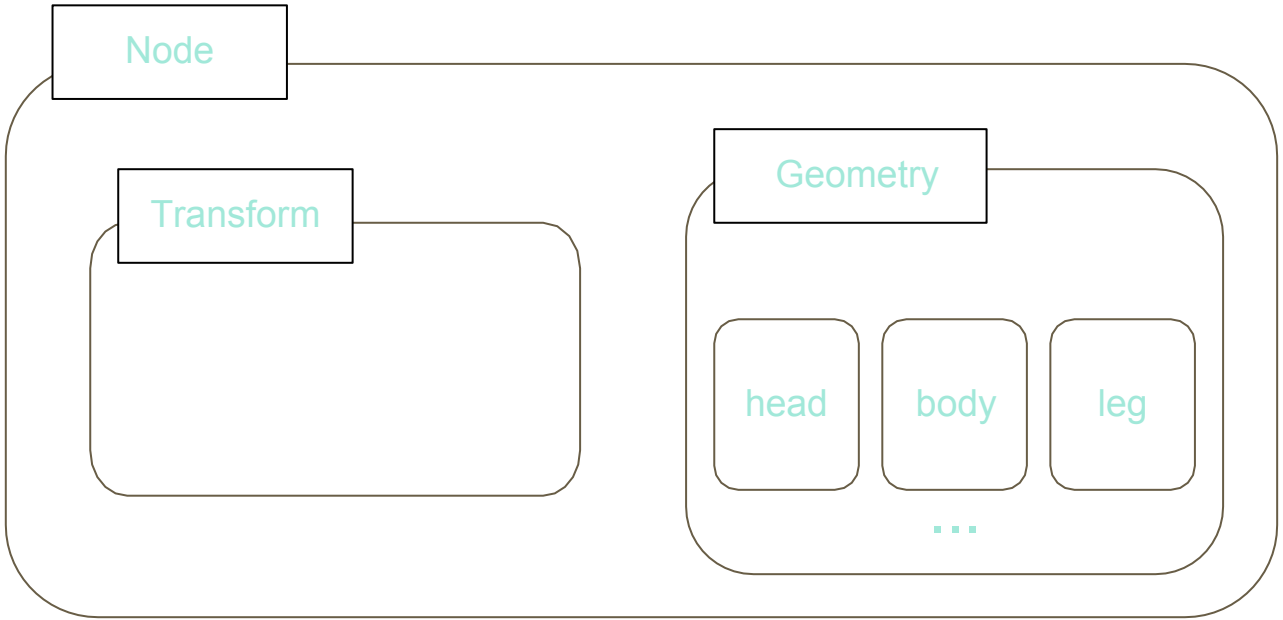
Texture: buffer and shader



Texture: buffer and shader

- A texture unit is of type sampler* in the fragment shader
  - Usually a sampler2D since we normally use 2D images as textures
  - This should be a uniform variable
    - uniform sampler2D *someVariableName*;
    - color = texture(someVariableName, TexCoords);
- The value of sampler2D is an unsigned int
  - *glUniform1i(...)*; or *glUniform1ui(...)*;
  - Value is what *textureID* is when we generate the texture

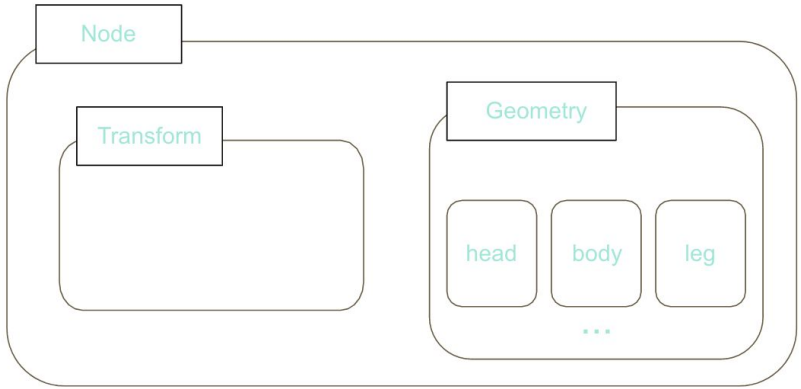
Scene graph: class hierarchy



Scene graph: class implementation

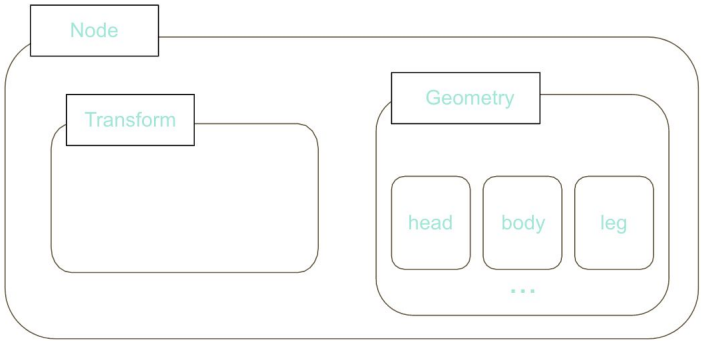
- Use virtual functions
- class Node

```
{
        …
        virtual void draw(int program, glm::mat4 M) = 0;
}

class MatrixTransform: public Node
{
        …
        void draw(int program, glm::mat4 M);

}
```

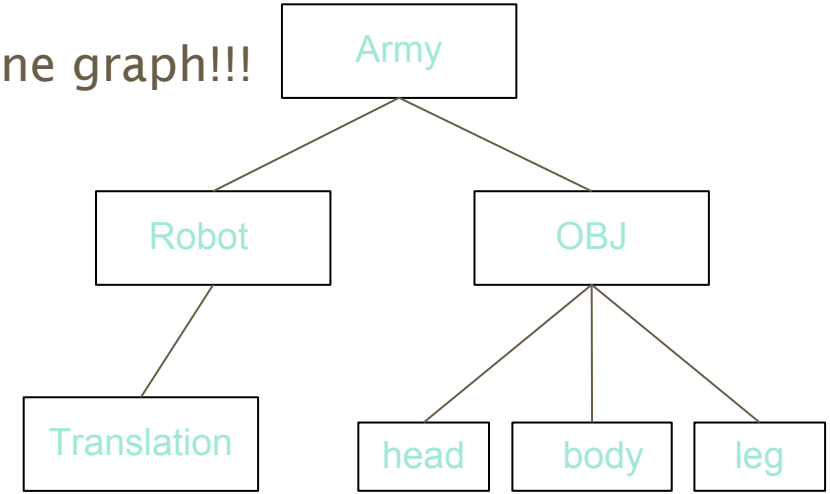


Scene graph: example

The class hierarchy is not equal to the scene graph!!!



≠

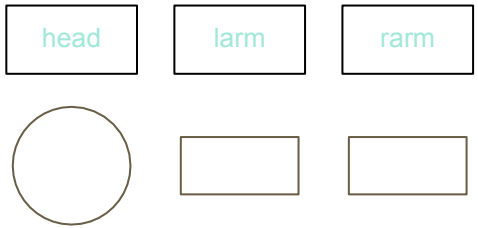


Scene graph: example(bottom-up)

Geometry* head = new OBJ(‘head’);

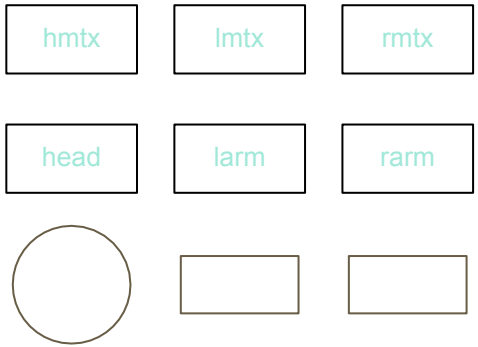
Geometry* larm = new OBJ(‘arm’);

Geometry* rarm = new OBJ(‘arm’);



Scene graph: example(bottom-up)

```
MT* hmtx = new MT(glm::mat4(1.0f));

MT* lmtx = new MT(glm::translate(glm::mat4(1.0f), vec3(-2.0f, 0.0f, 0.0f));

MT* rmtx = new MT(glm::translate(glm::mat4(1.0f), vec3(+2.0f, 0.0f, 0.0f));
```

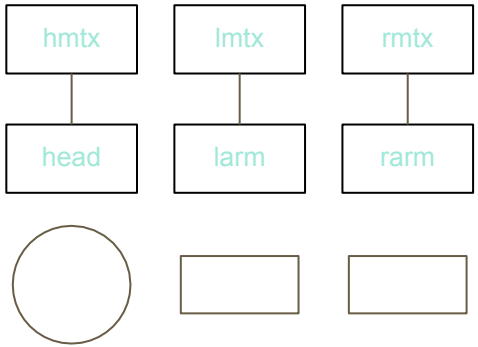


Scene graph: example(bottom-up)

hmtx->addChild(head);

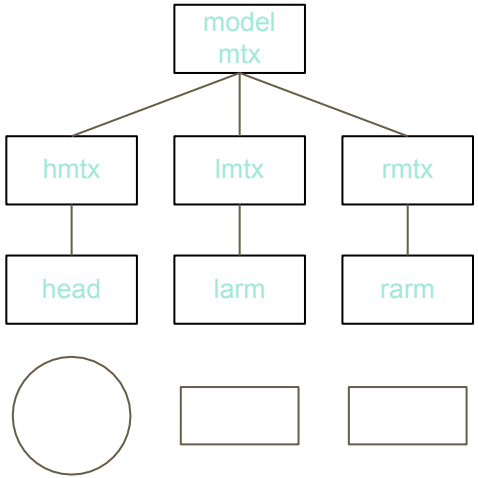
lmtx->addChild(larm);

rmtx->addChild(rarm);



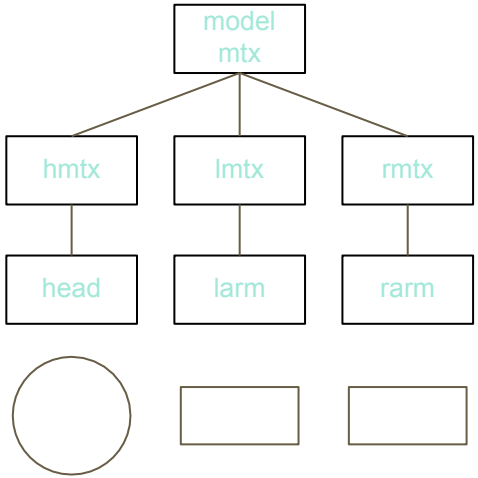
Scene graph: example(bottom-up)

```
MT* modelmtx = new MT(glm::mat4(1.0f));

modelmtx->addChild(hmtx);

modelmtx->addChild(lmtx);

modelmtx->addChild(rmtx);
```



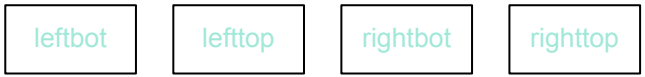
Scene graph: example(bottom-up)

If we call draw method on satellite right now, it would look like:



Scene graph: example(bottom-up)

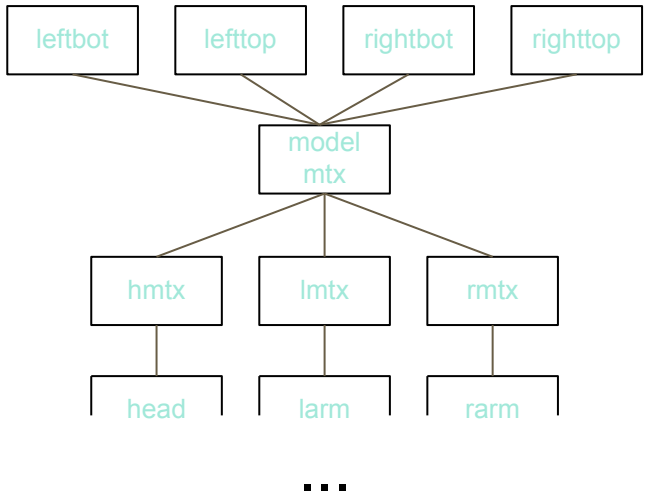
```
MT* leftbot = new MT(glm::translate(glm::mat4(1.0f), vec3(-1.0f, -1.0f, 0.0f));

MT* lefttop = new MT(glm::translate(glm::mat4(1.0f), vec3(-1.0f, +1.0f, 0.0f));

MT* rightbot = new MT(glm::translate(glm::mat4(1.0f), vec3(+1.0f, -1.0f, 0.0f));

MT* righttop = new MT(glm::translate(glm::mat4(1.0f), vec3(+1.0f, +1.0f, 0.0f));
```



Scene graph: example(bottom-up)

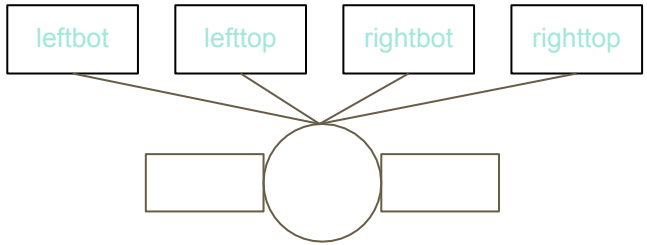
```
leftbot->addChild(modelmtx);

lefttop->addChild(modelmtx);

rightbot->addChild(modelmtx);

righttop->addChild(modelmtx);
```

Note that there is only a single instance of satellite!!!



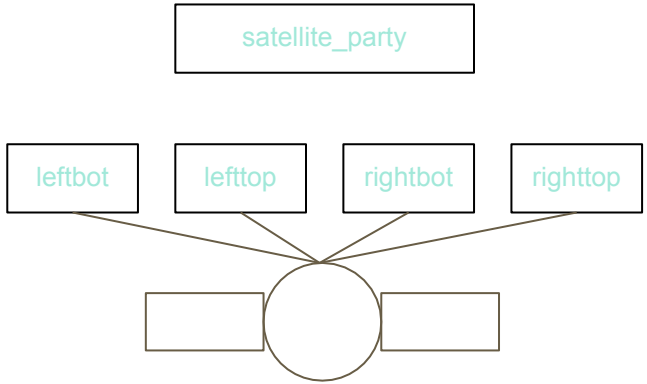
Scene graph: example(bottom-up)

```
leftbot->addChild(modelmtx);

lefttop->addChild(modelmtx);

rightbot->addChild(modelmtx);

righttop->addChild(modelmtx);
```



Scene graph: example(bottom-up)

MT* satellite_party = new MT(glm::mat4(1.0f));



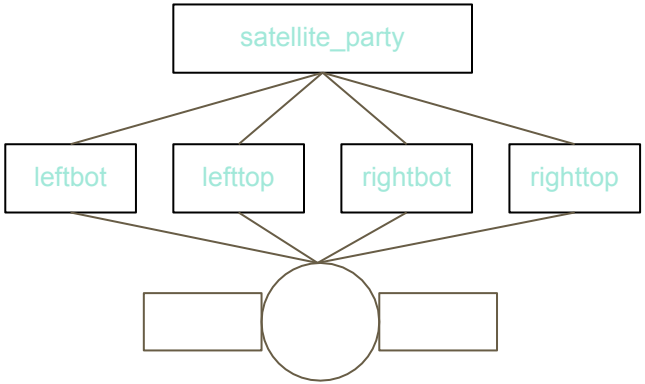
Scene graph: example(bottom-up)

satellite_party->addChild(leftbot);

satellite_party->addChild(lefttop);

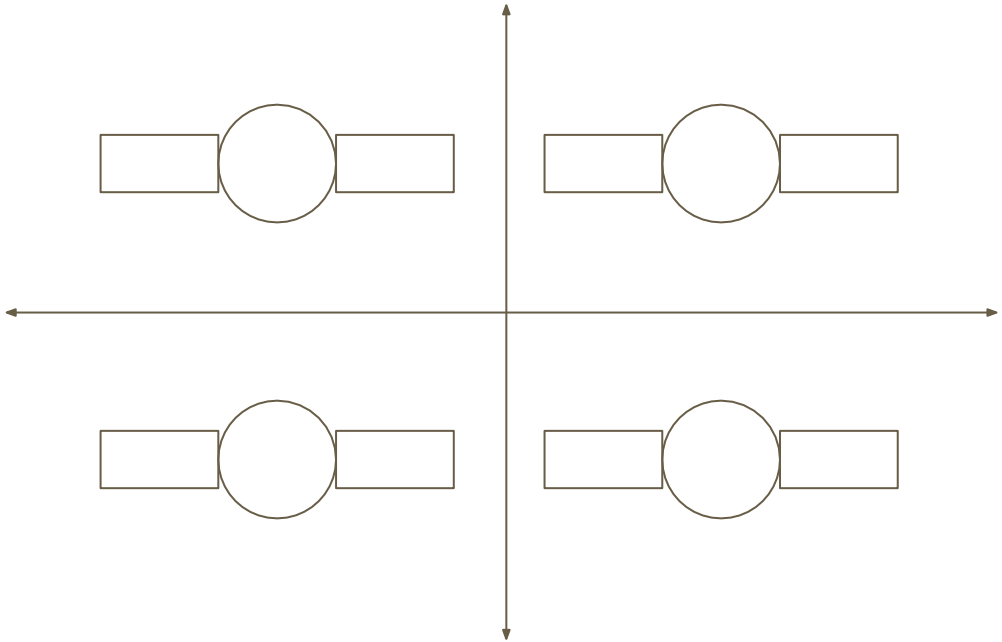
satellite_party->addChild(rightbot);

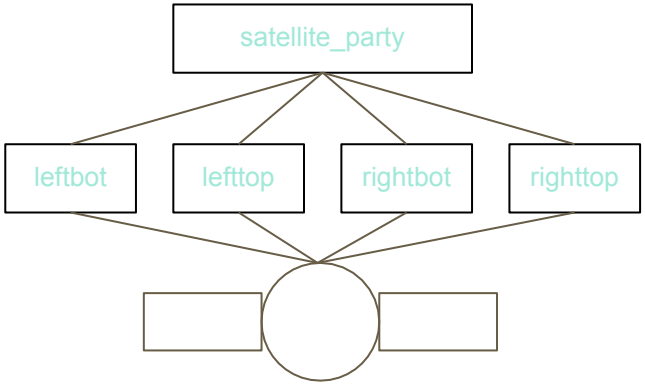
satellite_party->addChild(righttop);



Scene graph: example(bottom-up)

If we call draw method on satellite_party right now, it would look like:





Midterm Topics

- Linear Algebra
  - Basic vector properties (dot product, cross product)
  - Basic matrix properties (matrix multiplication, inverse, identity)

- Coordinate Systems
  - Homogeneous Coordinates
  - Scaling, Rotation, Translation
  - Model matrix, Camera matrix
    - Understand why the normal is not transformed directly by the model matrix

Midterm Topics

- Projection
  - Orthographic vs Perspective
  - Parameters for general/symmetric view volume (AKA frustum)

- “Complete Vertex Transformation in OpenGL”
  - Understand the whole series of transformations applied to go from a 3D model vertex to a 2D image position

Midterm Topics

- Illumination
  - Phong Illumination Model
    - Diffuse
    - Specular

- Light Source Properties
  - Directional
  - Point light
  - Spotlight

- Basic facts about lighting, e.g. Gouraud shading vs per-pixel shading, global illumination vs local (Phong) illumination

Midterm Topics

- Textures
  - Mapping, Interpolation
  - Wrapping
  - Texture coordinates AKA Surface parameterization
  - Anti-aliasing via Mipmaps

- Scene Graphs
  - Basically just applying data structures for organizing graphics