Adding New Features to the CS 5625 Framework

Pramook Khungurn

Understanding the CS 5625 framework

- Features
  - Lazy loading of data → loads data only when needed
  - Caching of data → no loading same file twice
  - GPU memory control → unused data automatically removed from GPU
- Mechanism: 3 layers of “data”
  - JOGL wrapper layer → classes that thinly wrap raw OpenGL objects
    - Vbo, Texture2D, TextureRect, TextureCubeMap
  - Data layer → raw data in main mem that can be turned into OpenGL objects
    - VertexData, IndexData, Texture2DData, TextureRectData, Mesh, Material
  - Object cache layer → provides lazy loading and caching
    - Holder, Reference, Value, ObjectCache

Object Cache Layer

- Feature: lazy loading + caching
- Principles
  - Objects rarely directly contain other objects.
  - Instead, they store “Holders” of other objects.
- Holder → 2 subclasses
  - Value → directly contains other objects.
  - Reference → contains a String called “key,” used to look up objects.
- ObjectCache contains the mapping from key → objects.

Data Layer

- All the data that are loaded from files / reside in main memory
  - Most objects are in this layer.
- Objects in data layer rarely directly contain other objects.
  - Store Holders to get lazy loading + caching
- 3 ways of object creation
  - Constructor
  - Loading from XML file → must implement XmlLoadable & XmlSavable
  - Loading from a key → Must provide an instance of ObjectLoader
- Some objects can “provide” OpenGL resources.
  - They implement the GLResourceProvider interface

JOGL Wrapper Layer

- Objects that deal with OpenGL and GPU data.
- Some are provided by objects in the Data layer:
  - Texture2D, Texture3D, TextureRect, Vbo, Program
- Some are convenience classes that wrap OpenGL calls:
  - Fbo, TextureUnit

New Features…

- New meshes
- New types of texture data
- New types of materials

- A new feature = new classes in the Data or JOGL layers.

New Mesh

New Mesh

- The framework can already deal with Wavefront OBJ format.
  - Especially those exported from Blender.

- Steps
  1. Export meshes from Blender.
  2. Create a scene that refers to the mesh.
     - Write a program to generate the XML scenes.
     - Create a WavefrontOBJTriMesh object that refers to the mesh file.
     - Put the mesh inside a SceneTreeNode
     - Save the SceneTreeNode to XML. (Scene #2)

New Mesh

```
public static void main(String[] args) {
    WavefrontOBJTriMesh mesh = new WavefrontOBJTriMesh(
            FileResolver.resolve("data/new_features/stdrect.obj"));

    SceneTreeNode root = new SceneTreeNode();

    SceneTreeNode meshNode = new SceneTreeNode();
    meshNode.setPosition(-0.5f, -0.5f, 0);
    meshNode.setData(new Value<>(mesh));
    root.addChild(meshNode);

    PointLight pointLight = new PointLight();
    pointLight.setPosition(0,0,10);
    pointLight.setQuadraticAttenuation(0.0f);
    SceneTreeNode lightNode = new SceneTreeNode();
    lightNode.setData(new Value<>(pointLight));
    root.addChild(lightNode);

    XmlUtil.saveXml(root, "data/new_features/rectangle_scene.xml");
}
```

Change Mesh Material

```
public static void main(String[] args) {
    WavefrontOBJTriMesh mesh = new WavefrontOBJTriMesh(
            FileResolver.resolve("data/new_features/stdrect.obj"));

    CustomizedSingleMaterialMesh customMesh = new CustomizedSingleMaterialMesh(new Value<>(mesh));
    LambertianMaterial material = new LambertianMaterial();
    material.setDiffuseColor(new Color4f(0.3f, 0.7f, 0.3f, 1.0f));
    customMesh.setMaterial(new Value<>(material));

    SceneTreeNode root = new SceneTreeNode();

    SceneTreeNode meshNode = new SceneTreeNode();
    meshNode.setPosition(-0.5f, -0.5f, 0);
    meshNode.setData(new Value<>(customMesh));
    root.addChild(meshNode);

        :
        :
        :

    XmlUtil.saveXml(root, "data/new_features/green_rectangle_scene.xml");
}
```

Change Mesh Material

```
public static void main(String[] args) {
    WavefrontOBJTriMesh mesh = new WavefrontOBJTriMesh(
            FileResolver.resolve("data/new_features/stdrect.obj"));

    CustomizedSingleMaterialMesh customMesh = new CustomizedSingleMaterialMesh(new Value<>(mesh));
    LambertianMaterial material = new LambertianMaterial();
    material.setDiffuseColor(new Color4f(0.3f, 0.7f, 0.3f, 1.0f));
    customMesh.setMaterial(new Value<>(material));

    SceneTreeNode root = new SceneTreeNode();

    SceneTreeNode meshNode = new SceneTreeNode();
    meshNode.setPosition(-0.5f, -0.5f, 0);
    meshNode.setData(new Value<>(customMesh));
    root.addChild(meshNode);

        :
        :
        :

    XmlUtil.saveXml(root, "data/new_features/green_rectangle_scene.xml");
}
```

Look at previous source code

- pa1.dataprep package
  - PrepareDefaultScene
  - PrepareManyLightScene
  - PrepareMaterialTestScene
  - PrepareNormalMappingTestScene

- Has examples on:
  - What to do if a mesh has many components.
  - How to change a mesh’s modeling transformation.

New Type of Texture Data

New Type of Texture Data

- Say, we want to load a new type of image.
  - Here, we use PFM file format, which was used in another course (CS 6630).
  - Say you write a class called “Pfm” that can read the file to main memory.
- Next, we need to create a new class in the Data layer.
  - Let us call it PfmTextureData.
  - Since it provides a texture data, it must implement Texture2DData.
  - We want to load it with a reference to the PFM file name.
    - So, we must provide an instance of ObjectLoader.
    - We also have to register the loader every time before we use it.

PfmTextureData

```
public class PfmTextureData implements Texture2DData {
    public static String PROTOCOL_NAME = "pfmTextureData";

    private static Logger logger = LoggerFactory.getLogger(PfmTextureData.class);
    String fileName;
    Pfm pfm;
    int version = 1;

    PfmTextureData(String fileName) {
        this.fileName = fileName;
        pfm = Pfm.load(fileName);
    }

    @Override
    public int getWidth() {
        return pfm.width;
    }

    @Override
    public int getHeight() {
        return pfm.height;
    }
```

GLResourceProvider<T>

- Interface for objects that provide OpenGL “resource”.
  - Resource = object in the JOGL wrapper layer.
- Two methods
  - updateGLResource(GL2 gl, GLResourceRecord record)
    - Update the OpenGL object inside the record.
    - Must update 3 things:
      - The OpenGL resource itself.
      - The size of the object in bytes.
      - The version of the object.
  - getGLResource(GL2 gl)
    - Convenience method for fetching object from the cache.
- Texture2DData = GLResourceProvider<Texture2D> + extra methods.

PfmTextureData (cont.)

- For getGLResource(GL2 gl), just retrieve the object from cache.

```
@Override
public Texture2D getGLResource(GL2 gl) {
    return (Texture2D) GLResourceCache.v().getGLResource(gl, this);
}
```

PfmTextureData (cont.)

- For updateGLResource
  - Check whether you need an update
    - If the resource in the record is null.
    - The version is not equal to the local version of the object.
  - When updating:
    - Assign the resource.
    - Assign the new version.
    - Assign the size in bytes.

PfmTextureData (cont.)

```
@Override
public void updateGLResource(GL2 gl, GLResourceRecord record) {
    Texture2D texture = null;
    boolean needUpdate = false;
    if (record.resource == null) {
        texture = new Texture2D(gl);
        record.resource = texture;
        needUpdate = true;
    } else if (record.version != this.version) {
        texture = (Texture2D) record.resource;
        needUpdate = true;
    }
    if (needUpdate) {
        logger.debug("texture file name = " + fileName);
        updateTexture2D(texture);
        record.version = version;
        record.sizeInBytes = pfm.width * pfm.height * 4;
    }
}
```

PfmTextureData (cont.)

```
@Override
public void updateTexture2D(Texture2D texture) {
    ByteBuffer buffer = ByteBuffer.allocate(getWidth() * getHeight() * 4 * 4);
    Color3d color = new Color3d();
    for (int y = 0; y < getHeight(); y++) {
        for (int x = 0; x < getWidth(); x++) {
            pfm.getColor(x, y, color);
            BufferUtil.setLittleEndianFloat(buffer, 4*(y*getWidth() + x), (float)color.x);
            BufferUtil.setLittleEndianFloat(buffer, 4*(y*getWidth() + x) + 1, (float)color.y);
            BufferUtil.setLittleEndianFloat(buffer, 4*(y*getWidth() + x) + 2, (float)color.z);
            BufferUtil.setLittleEndianFloat(buffer, 4*(y*getWidth() + x) + 3, 1.0f);
        }
    }
    buffer.rewind();
    texture.setImage(getWidth(), getHeight(), GL2.GL_RGBA, GL2.GL_FLOAT, buffer);
}
```

ObjectLoader

- Allows the framework to load the object from a keyword through a Reference.
  - <ref func="diffuseTexture" key="pfmTextureData||||envmap_00.pfm"/>
- 3 methods:
  - resolveKey
    - Resolve any file names inside a key at loading time.
  - relativizeKey
    - Relativize any file names inside a key at saving time.
  - load
    - Create an object from a key.

PfmTextureData (cont.)

```
public class PfmTextureData implements Texture2DData {

    public static class Loader implements ObjectLoader {
        @Override
        public String resolveKey(String key) {
            String keyData = ObjectCacheKey.getKeyData(key);
            String resolvedKeyData = FileResolver.resolve(keyData);
            return ObjectCacheKey.makeKey(PROTOCOL_NAME, resolvedKeyData);
        }

        @Override
        public String relativizeKey(String key) {
            String keyData = ObjectCacheKey.getKeyData(key);
            String relativeKeyData = FileResolver.relativize(keyData);
            return ObjectCacheKey.makeKey(PROTOCOL_NAME, relativeKeyData);
        }

        @Override
        public Object load(String key) {
            PfmTextureData result = new PfmTextureData(ObjectCacheKey.getKeyData(key));
            return result;
        }
    }
}
```

Creating scene with new PfmTextureData

```
public static void main(String[] args) {
    ObjectLoader.registerLoader(PfmTextureData.PROTOCOL_NAME, new PfmTextureData.Loader());

    WavefrontOBJTriMesh mesh = new WavefrontOBJTriMesh(
            FileResolver.resolve("data/new_features/stdrect.obj"));

    Reference textureRef = new Reference(ObjectCacheKey.makeKey(PfmTextureData.PROTOCOL_NAME,
            "data/new_features/envmap_00.pfm"));

    LambertianMaterial material = new LambertianMaterial();
    material.setDiffuseColor(new Color4f(1.0f, 1.0f, 1.0f, 1.0f));
    material.setDiffuseTexture(textureRef);

    CustomizedSingleMaterialMesh customMesh = new CustomizedSingleMaterialMesh(new Value<>(mesh));
    customMesh.setMaterial(new Value<>(material));

        :
        :
        :

    XmlUtil.saveXml(root, "data/new_features/pfm_rectangle_scene.xml");
}
```

Creating scene with new PfmTextureData

```
public static void main(String[] args) {
    ObjectLoader.registerLoader(PfmTextureData.PROTOCOL_NAME, new PfmTextureData.Loader());

    WavefrontOBJTriMesh mesh = new WavefrontOBJTriMesh(
            FileResolver.resolve("data/new_features/stdrect.obj"));

    Reference textureRef = new Reference(ObjectCacheKey.makeKey(PfmTextureData.PROTOCOL_NAME,
            "data/new_features/envmap_00.pfm"));

    LambertianMaterial material = new LambertianMaterial();
    material.setDiffuseColor(new Color4f(1.0f, 1.0f, 1.0f, 1.0f));
    material.setDiffuseTexture(textureRef);

    CustomizedSingleMaterialMesh customMesh = new CustomizedSingleMaterialMesh(new Value<>(mesh));
    customMesh.setMaterial(new Value<>(material));

        :
        :
        :

    XmlUtil.saveXml(root, "data/new_features/pfm_rectangle_scene.xml");
}
```

Also, in your main program…

```
public class NewFeatures extends JFrame implements GLController, ActionListener {
    public static void main(String[] args) {
        ObjectLoader.registerLoader(PfmTextureData.PROTOCOL_NAME, new PfmTextureData.Loader());

        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception e) {
            // NOP
        }
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new NewFeatures().run();
            }
        });
    }
}
```

New Material

New Material

- 3 steps
  - Write material class.
  - Write shaders.
  - Modify the renderer.
- Write material class.
  - Data only class. Should not contain any code that deals with rendering.
  - Concentrate on:
    - What data is needed for this material.
    - How to serialize.
  - Must implement XmlLoadable & XmlSavable.
  - Create XmlBuilder inner class and register it.
  - Requirement: Must have a default constructor!

MonochromeMaterial

```
public class MonochromeMaterial extends AbstractMaterial implements XmlLoadable, XmlSavable {
    public static final String TYPE_NAME = "monochromeMaterial";

    private Color3f color = new Color3f();
    private Holder<Texture2DData> texture;

    public Color3f getColor() {
        return color;
    }

    public Holder<Texture2DData> getTexture() {
        return texture;
    }
```

XmlLoadable

- Implement this interface to make a class loadable from XML file
- 3 methods
  - setXmlProperties
    - Deals with int/float/Color3f/Point3f “properties” specified in XML file.
  - addXmlChild
    - Deals with child objects inside an object’s XML definition.
  - activateXml
    - Called after the instance have been created with all properties and children.
    - It should check the validity of the instance’s parameters.

MonochromeMaterial (cont.)

```
public class MonochromeMaterial extends AbstractMaterial implements XmlLoadable, XmlSavable {
    @Override
    public void setXmlProperties(HashMap<String, Object> properties) {
        setBlendingProperties(properties);
        color.set(PropertiesUtil.getColor3f(properties, "color", new Color3f(1.0f, 1.0f, 1.0f)));
    }

    @Override
    public void addXmlChild(String func, Holder child) {
        if (func.equals("texture")) {
            if (texture != null) {
                throw new RuntimeException("Attempted to assign texture twice!");
            } else {
                texture = child;
            }
        }
    }

    @Override
    public void activateXml() {
        if (texture == null) {
            throw new RuntimeException("texture has not been assigned!");
        }
    }
}
```

XmlSavable

- Implement this class so that you can serialize to XML
- Only one method: toXml

```
public class MonochromeMaterial {
    @Override
    public Element toXml(Document document) {
        Element result = XmlUtil.createObjectElement(document, TYPE_NAME);
        XmlUtil.addPropertyElement(result, "color", color);
        result.appendChild(XmlUtil.toXml(document, "texture", texture));
        return result;
    }
}
```

XmlBuilder

- Create a static inner class extending cs5625.gfx.load.xml.XmlBuilder
- Register this class before you use it in your program.

```
public class MonochromeMaterial {
    public static class XmlBuilder extends cs5625.gfx.loading.xml.XmlBuilder {
        @Override
        protected XmlLoadable createInstance() {
            return new MonochromeMaterial();
        }
    }
}
```

Write the shader.

- For a material, only a fragment shader is needed.
  - The vertex shader is determined by the mesh type.

```
#version 120

varying vec2 geom_texCoord;
uniform sampler2D texture;
uniform vec3 color;

float luminance(vec3 c) {
    return c.x * 0.212671 + c.y * 0.715160 + c.z * 0.072169;
}

void main() {
    vec4 texValue = texture2D(texture, geom_texCoord);
    float l = luminance(texValue.xyz);
    glFragColor = vec4(color*l, 1.0);
}
```

Modify the renderer

- In this example, ForwardMeshRenderer deals with shader invocation.
- 2 steps
  - Add a branch to renderMeshPart.
  - Write a function that deals with MonochromeMaterial
- Same as in PA1 and PA6.

Create the scene…

```
public class PrepareMonochromeRectangleScene {
    public static void main(String[] args) {
        ObjectLoader.registerLoader(PfmTextureData.PROTOCOL_NAME, new PfmTextureData.Loader());
        XmlBuilder.registerBuilder(MonochromeMaterial.TYPE_NAME, MonochromeMaterial.XmlBuilder.class);

        WavefrontOBJTriMesh mesh = new WavefrontOBJTriMesh(
                FileResolver.resolve("data/new_features/stdrect.obj"));

        CustomizedSingleMaterialMesh customMesh = new CustomizedSingleMaterialMesh(new Value<>(mesh));
        MonochromeMaterial material = new MonochromeMaterial();
        material.setColor(new Color3f(0.3f, 0.7f, 0.3f));
        Reference textureRef = new Reference(ObjectCacheKey.makeKey(PfmTextureData.PROTOCOL_NAME,
                "data/new_features/envmap_00.pfm"));
        material.setTexture(textureRef);
        customMesh.setMaterial(new Value<>(material));

            :
            :
            :

        XmlUtil.saveXml(root, "data/new_features/monochrome_rectangle_scene.xml");
    }
}
```

Create the scene…

```
public class PrepareMonochromeRectangleScene {
    public static void main(String[] args) {
        ObjectLoader.registerLoader(PfmTextureData.PROTOCOL_NAME, new PfmTextureData.Loader());
        XmlBuilder.registerBuilder(MonochromeMaterial.TYPE_NAME, MonochromeMaterial.XmlBuilder.class);

        WavefrontOBJTriMesh mesh = new WavefrontOBJTriMesh(
                FileResolver.resolve("data/new_features/stdrect.obj"));

        CustomizedSingleMaterialMesh customMesh = new CustomizedSingleMaterialMesh(new Value<>(mesh));
        MonochromeMaterial material = new MonochromeMaterial();
        material.setColor(new Color3f(0.3f, 0.7f, 0.3f));
        Reference textureRef = new Reference(ObjectCacheKey.makeKey(PfmTextureData.PROTOCOL_NAME,
                "data/new_features/envmap_00.pfm"));
        material.setTexture(textureRef);
        customMesh.setMaterial(new Value<>(material));

            :
            :
            :

        XmlUtil.saveXml(root, "data/new_features/monochrome_rectangle_scene.xml");
    }
}
```