CuRLI_v1

CS6610 Interactive Computer Graphics HW repository. Stands for **C**omp**U**ter **R**enders **L**ot of **I**mages. It is inteded as a toy renderer/course-project.

Note to TAs: If you are viewing this as a PDF it is the same document as the ReadMe.md so you may use that if it is more convinient.

Building CuRLI for Windows

Building CuRLI for windows requires Cmake 3.0

Dependencies

Most of these dependencies are included as submodules and compiles with CMake. Only dependency that does not exist as submodule is glad which can be downloaded from generated link obtained glad web page.

- MSVCv143 - VS2022 C++ or above
- Cmake 3.0
- GLM
- ImGui
- GLFW ~~FreeGlut~~
- Glad
- cyCodeBase(only for .obj importer ATM)
- EnTT

Building with Cmake

1. Clone or download the files(unzip the downloaded files).
2. Create a folder to *build* binaries.
3. Run CmakeGui or Cmake select source as the project root directory and where to build binaries as *build* folder.
4. Select Visual Studio 17 2022 as the generator.
5. Configure and Generate
6. Navigate to *build* folder and open curli.sln file with Visual Studio

7. Select Under Build>Build solution(F7)
8. Run the executable i.e. **./curli.exe [params]** from console

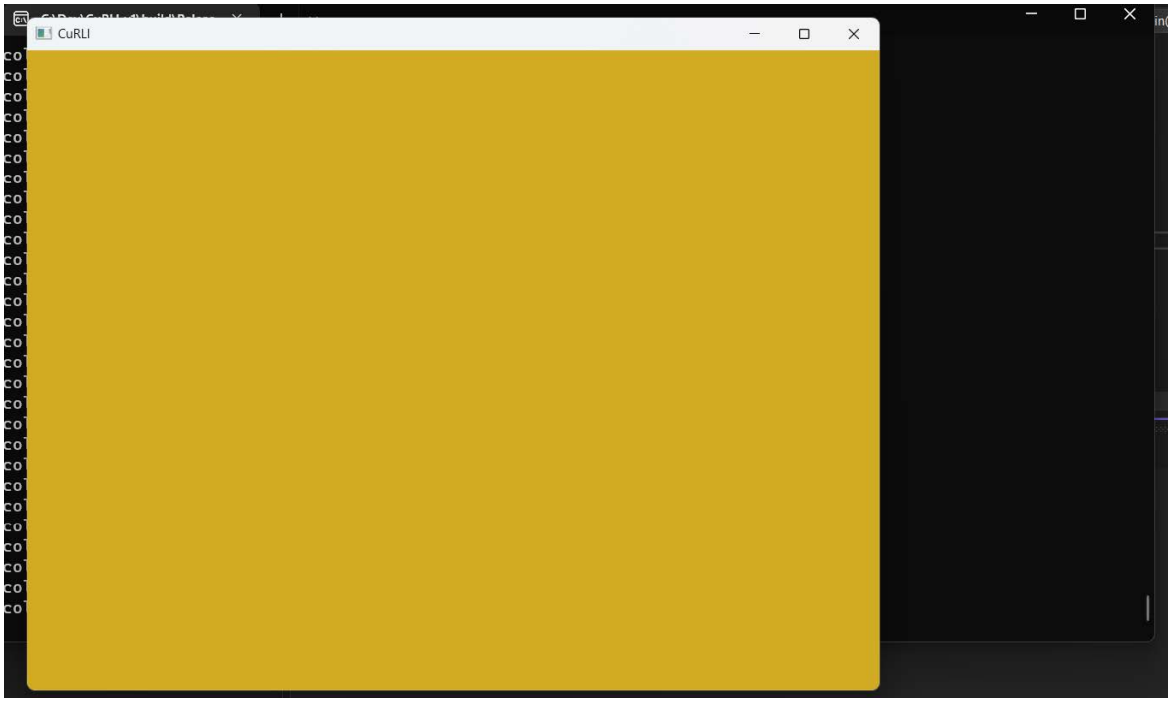
Milestones

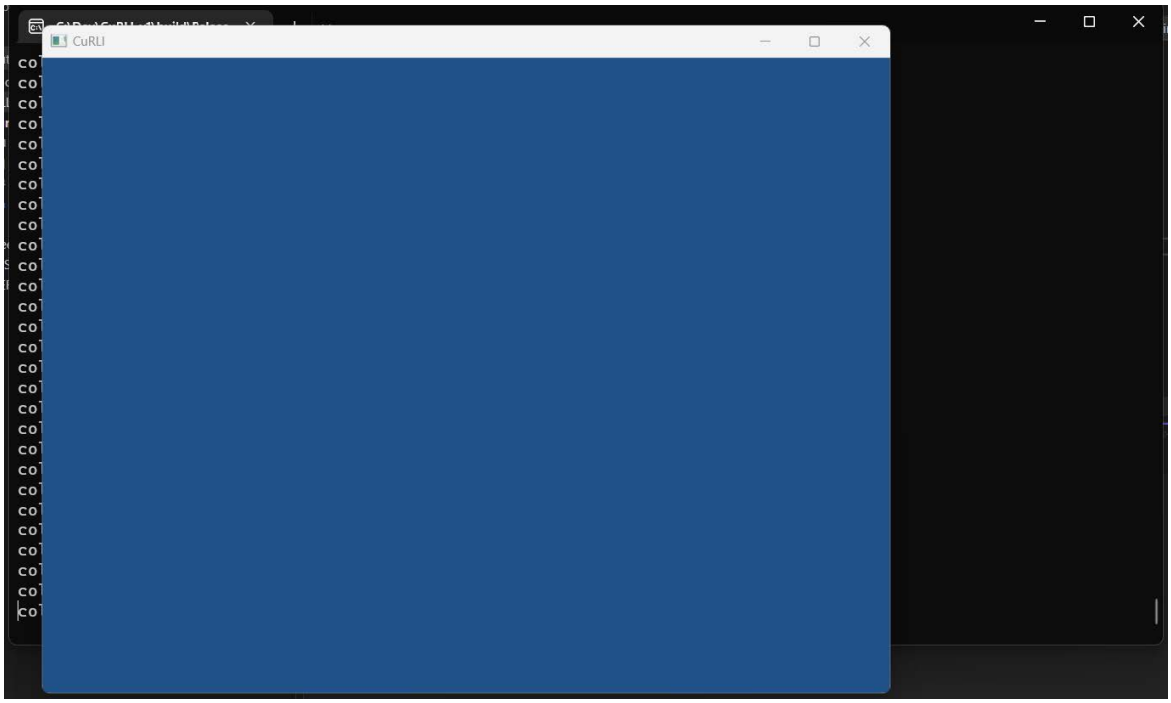
Project 1 - H*llo World

Project 1 requirements:

- [x] Creating Window context
- [x] Keyboard listeners where ‘esc’ is used to call `glutLeaveMainLoop();`
- [x] Setting window size, position, name and clear color during initialization.
- [x] Idle function where animation between two colors are generated using linear interpolation of sine value of time(ms).

Some Screenshots:





Project 2 - Transformations

Project 2 requirements:

- [x] Integrated `cyTriMesh` class to load .obj files from console arguments. Now path to a .obj mesh needs to be given to executable as the first argument as follows: `./curli.exe path/to/mesh`
- [x] Implemented very simple shaders (/assets/shaders/simple/...) to transform and render vertex points in a constant single color as `GL_POINTS`.
- [x] Implemented tarball controlled lookAt camera where `left mouse button + drag` adjusts two angles of the camera and `right mouse button + drag` adjusts the distance of the camera to *center*.
- [x] Programmed a Imgui window and keyboard shortcuts that allows reloading(`F5`) and recompiling(`F6`) of the shader files. This means that one can edit shader files after curli launches, pressing F5 and F6 will use the edited shaders if compilation is successful.
- [x] Imgui window also includes a button that recenters camera(`F1`) to the mesh center point.
- [x] Pressing `P` also lets user switch between orthographic and perspective projection types.

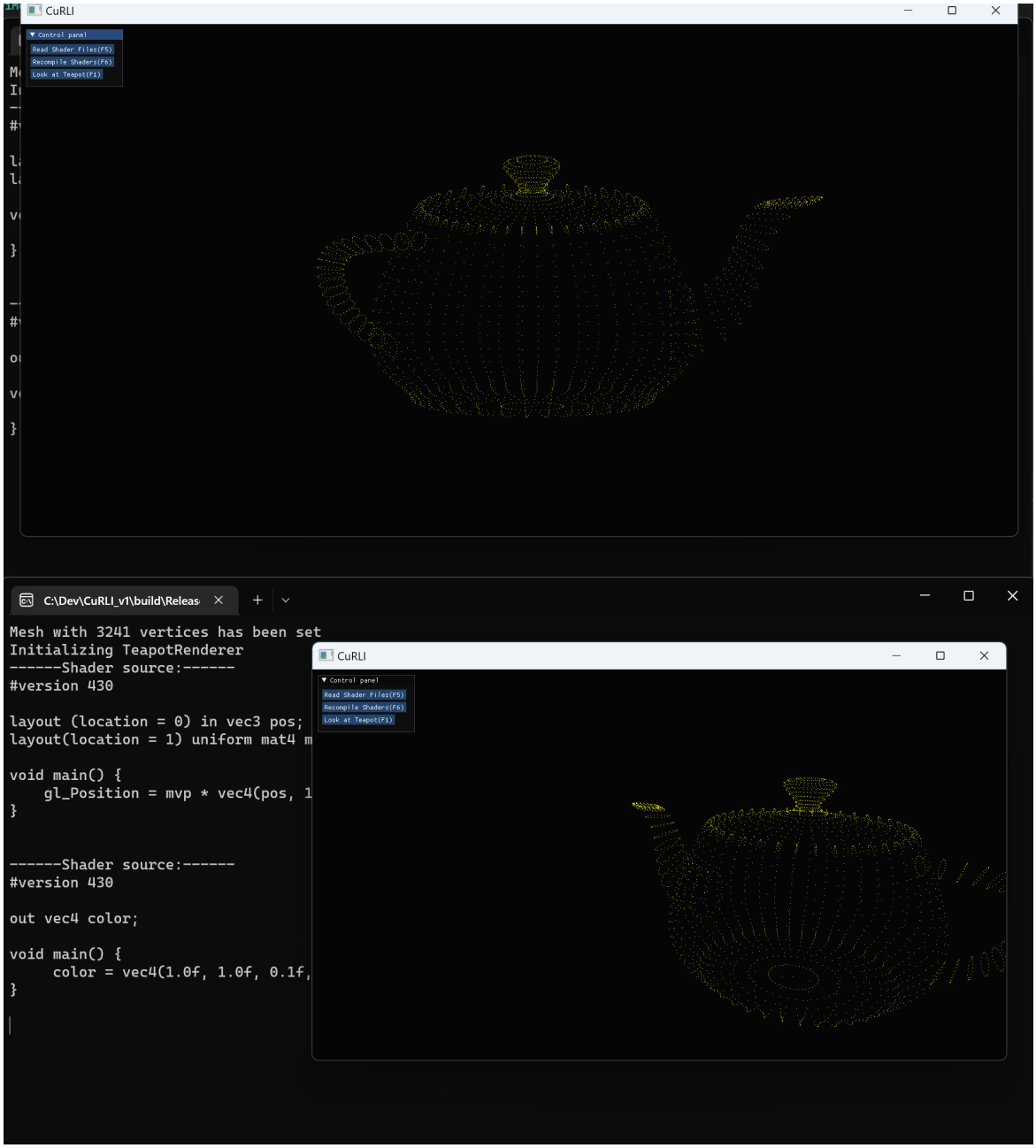
Additional Features:

- Curiously recurring template pattern has been utulized to have staged renderers and application.
  - Application stages are: `Initialize()`, `Render()`, `DrawGui()`, `Terminate()`.Render and DrawGui are called in a render loop.
  - Each renderer `A` that implements base class `Renderer<A>` will need to override `Start()` `PreUpdate()` `Update()` and `End()`. These functions are called on various stages of the

application allowing custimizable renderers to be written.

- Also programmed a simple event dispatcher system which gets the input&windowing events by `glfw` to be queued. The queued event is resolved in render loop.
  - Each renderer has the option to override certain event calls dispatched by the system if the fuctions are overriden they are called by the `dispatchEvent()`
- Inside `PreUpdate()` function of this projects renderer (`TeapotRenderer`) I set the model matrix of the teapot to a rotation matrix that updates the angle over time. This causes teapot to revolve around itself.

Some Screenshots:



Project 3 - Shading

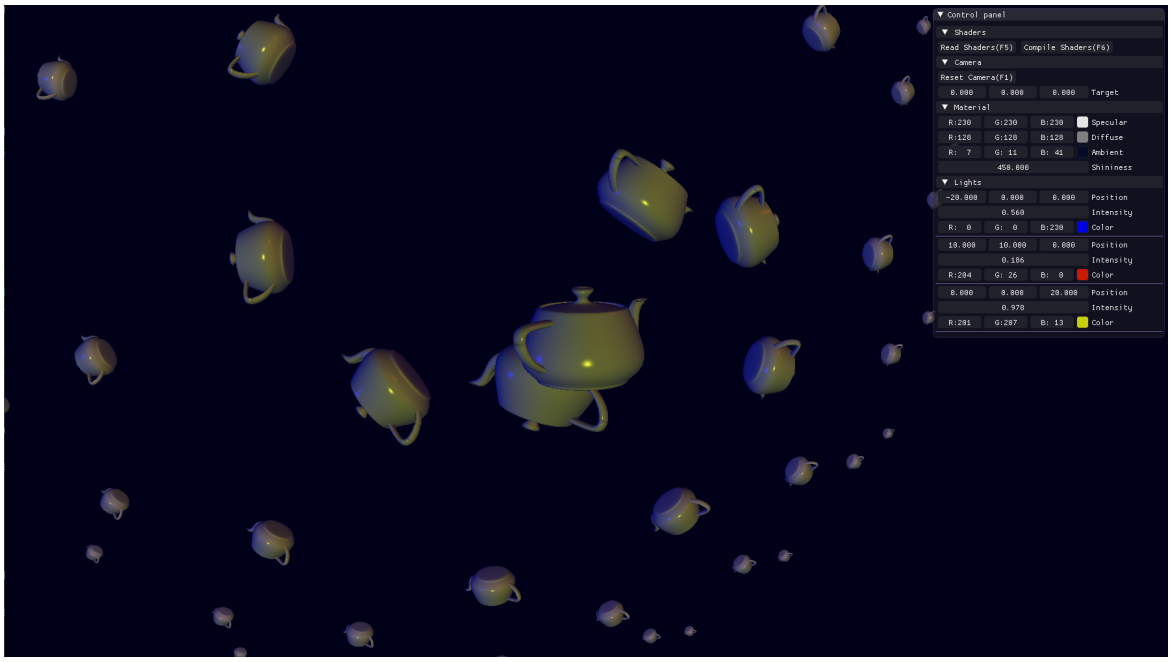
Project 3 requirements:

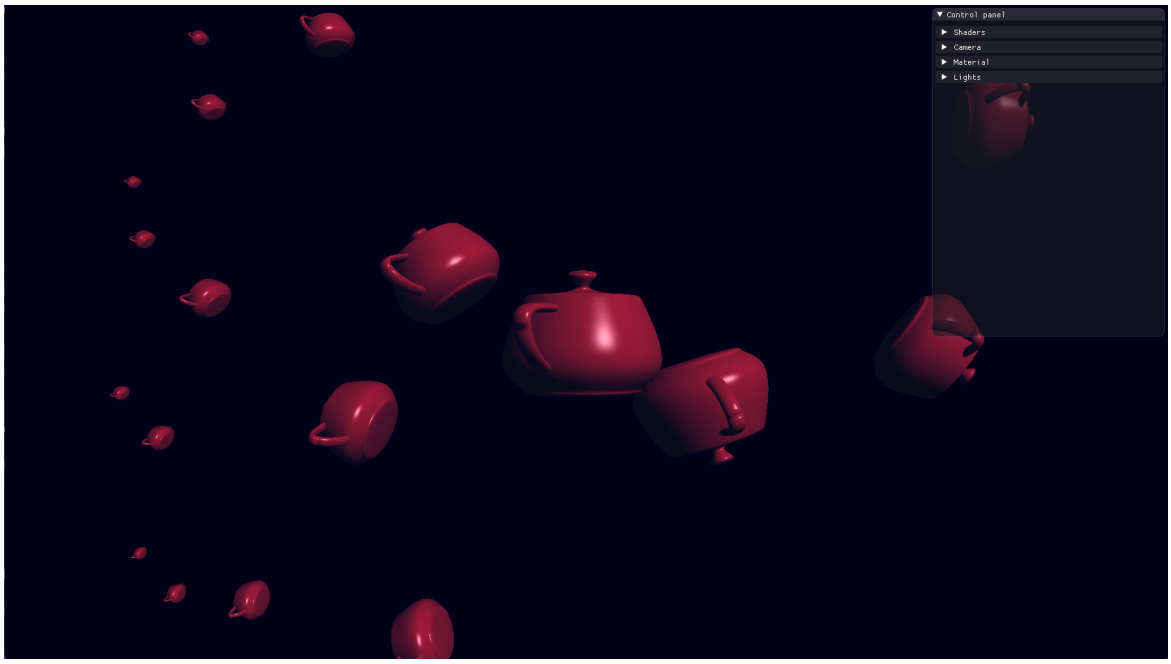
- [x] Displayed triangles instead of points
- [x] Uploaded and transforming vertex normals using inverse transpose of model view matrix
- [x] Imlemented Blinn-Phong shading in view space using half angles.
- [x] Added orbital controls to the first point light source inserted into the scene.

Additional Features:

- Integrated EnTT --- an entity-component system.
  - Using EnTT several components have been developed:
    - `CLight` : Illuminates the scene currently only as point light but soon other types will be implemented.
    - `CTransform` : A transform component that is traditionally used to generate model matrices for shaders
    - `CTriMesh` : Wrapper for cyTriMesh allows entities to have geometry
    - `CVertexArrayObject` : Allows geometry to be drawn using bound VBOs and EBO(optional). Automaticly handles and selects which draw calls to make.
- Implemented `OpenGLProgram` abstraction which alows convenience binding shaders and uploading uniforms.
- Using the ECS I let shaders render multiple light sources over multiple objects.

Some Screenshots:



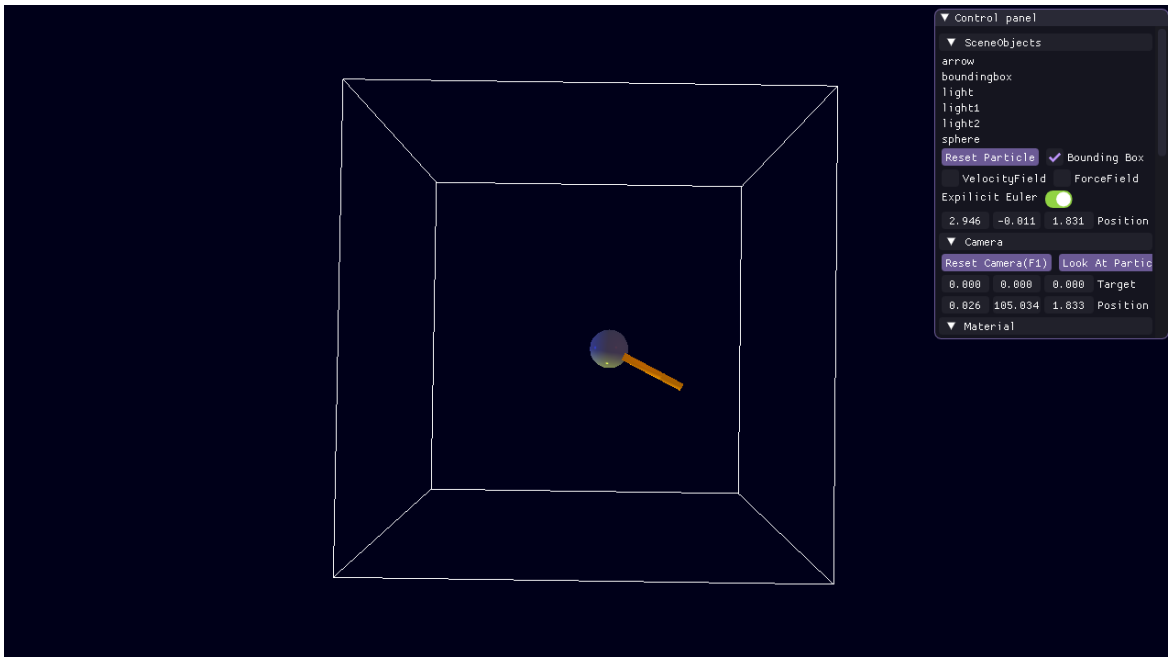


Physically Based Animation - Assignment 1

A simple one-particle system:

- [x] Rendering and shading 3D sphere as a particle.
- [x] `LShift + Mouse drag` can interactively apply force.
- [x] 3D arrow as force vector indicator.
- [x] Programmable VelocityField2D component with respective UI to add/remove from the scene.
- [x] Programmable ForceField2D component with respective UI to add/remove from the scene.
- [x] Euler integrator (explicit or implicit) and its selection UI.
- [x] BoundingBox class with VertexArrayObject component so it can be drawn as GL_LINES.

Some Screenshots:



Project 4 - Textures

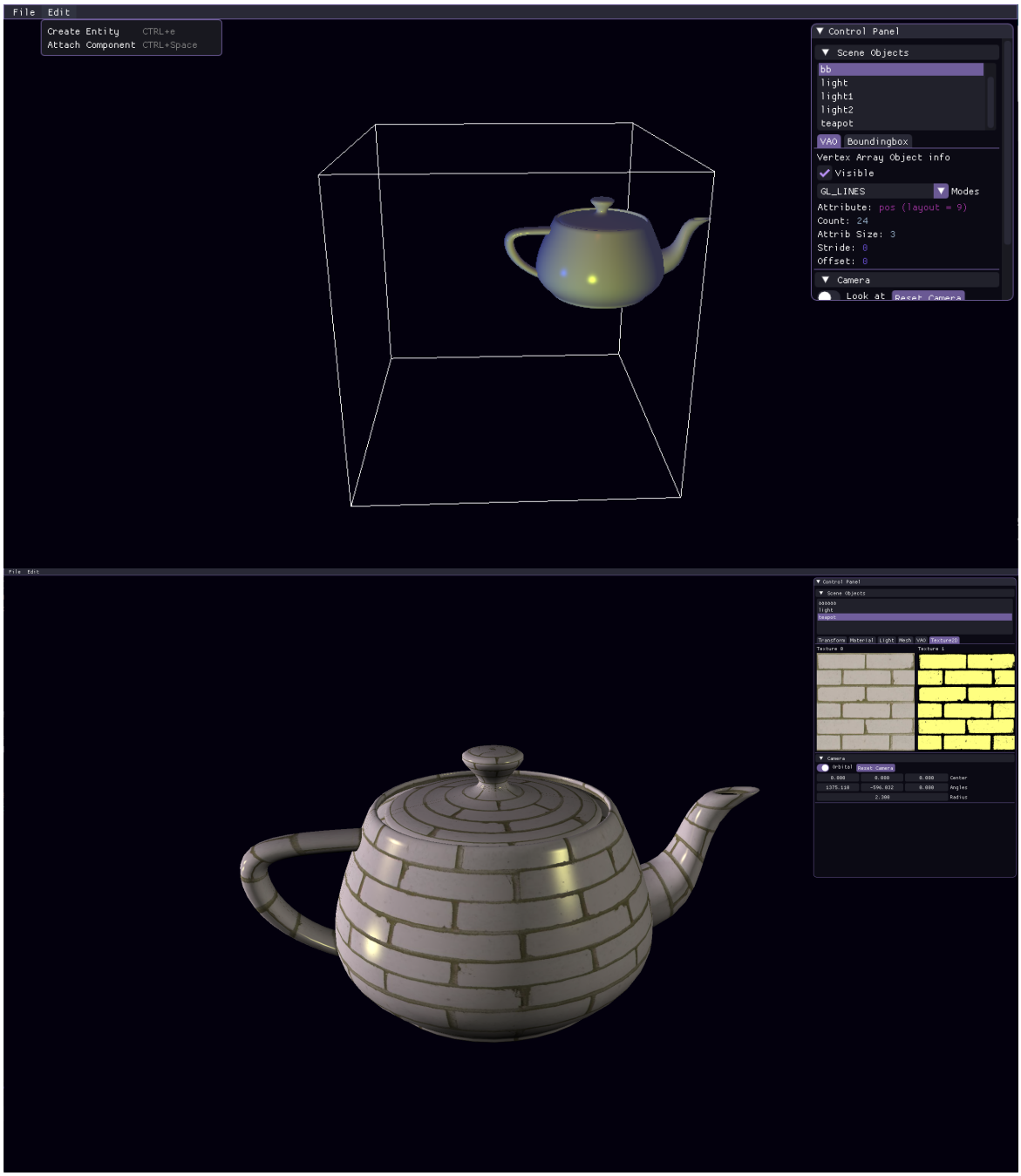
Project 4 requirements:

- [x] Load and parse .mtl files associated with .obj files.
- [x] Load and decode .png files as textures.
- [x] Use cmdline to read .obj files
    - Usage `-model --path ../path/to/your.obj`
    - Appending `--rb` after `-model` tag attaches `CRigidBody` component to the model which includes this object in WIP physics events.
- [x] Display textures properly on the object.
- [x] Include the specular texture, specified in the mtl file, for adjusting the specular color of the object.

Additional Features:

- Improved UI using ImGUI and EnTT.
    - Now scene objects are displayed in a list.
    - Selecting an object from a list allows user to see different components attached to the object.
- Using the Top menu bar user can load .obj files.
    - `File>import .obj file`
- Again using the top menu bar user can add components to the selected scene object.
    - Under `Edit > Attach component`
- User can also create empty entities to attach objects to it using `Edit>Create Entity`

Some Screenshots:



Project 5 - Render Buffers

Project 5 requirements:

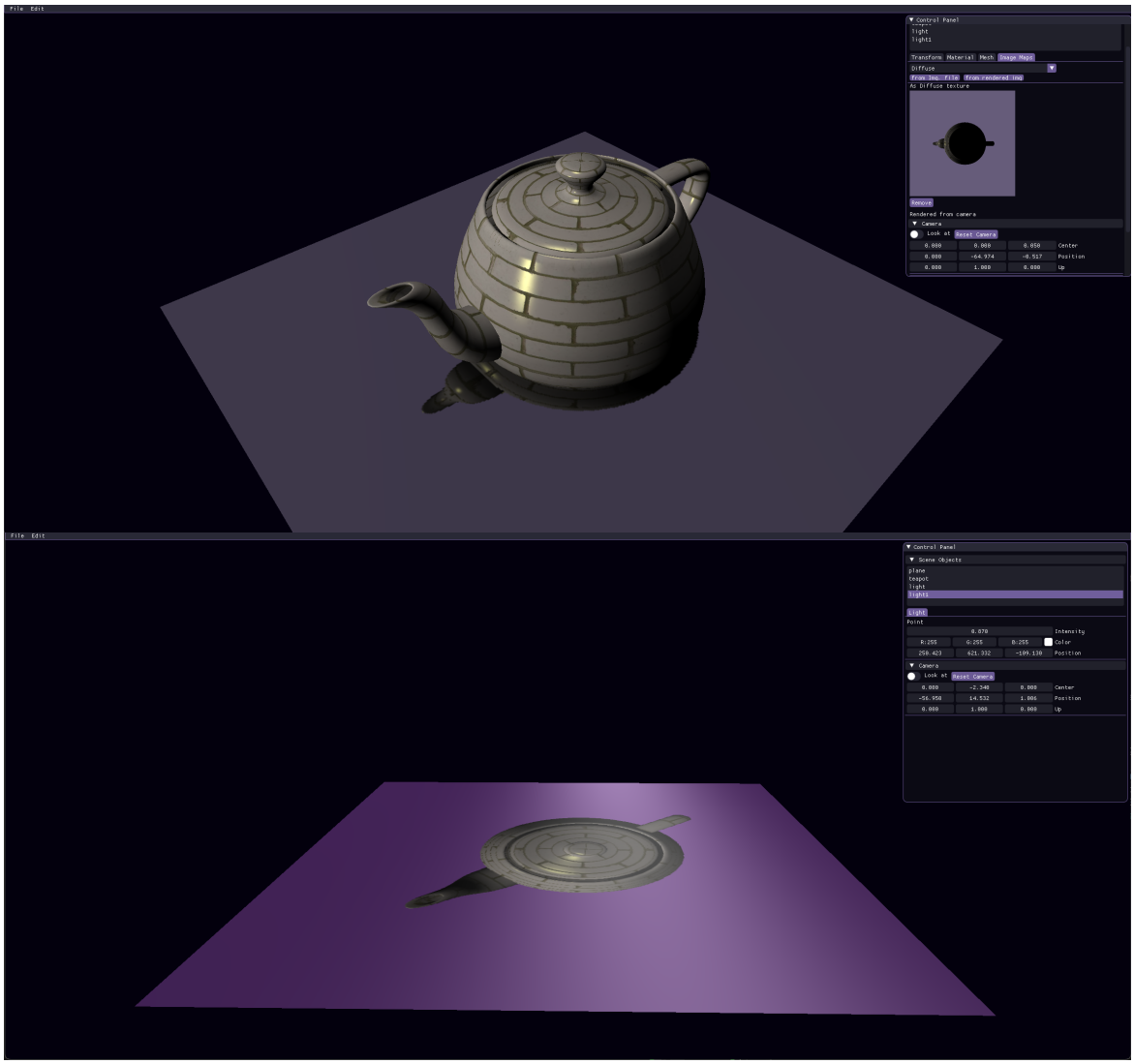
- [x] Loads and parses `.obj` files as command-line arguments.
  - Usage `-model --path ../path/to/your.obj`
  - For this specific example to work call the program with a plane.obj file provided inside assets. I have set up a scene for easy grading
- [x] The objects are rendered with the textures coupled with their `.obj` and `.mtl` files.
- [x] The provided scene can be rendered without the teapot on the actual viewport (but on the texture) by adjusting visibilty settings on Mesh component using the GUI.
- [x] Display the rendered texture by mapping it on a square-shaped plane.

- [x] Camera controls work the same as with the previous assignments.
- [x] If the ALT key is pressed, the left and right mouse buttons (and drag) controls the same view parameters for rendering texture on the plane.
- [x] Background of the image plane is set to phong diffuse color of the plane to separate it from the background color.
- [x] The rendered texture uses bilinear filtering for magnification and mip-mapping with anisotropic filtering for minification.

Additional Features:

- Seperated OpenGL concepts like `VertexArrayObject` and `Textures` are seperated from scene/entity component system.
  - This seperation allows better and cleaner implementation.
  - For textures I have implemented a `CImageMaps` component which maintains a `std::vector` of `ImageMap`. `ImageMap` data and properties are converted to `OpenGLProgram` Textures and bound acordingly.
  - For Rendered textures I maintain a wrapper struct for `Texture2D` struct which creates frame and depth buffers on request. This struct comes with a `Render(...)` function. This function takes another `std::function` as parameter and calls it after binding the relevant buffers and the viewport. Calling `RenderedTexture2D` 's Render function at Renderer's `Preupdate` function I can pass the `Update` function pointer as a parameter and render the scene as it would to that texture.
  - The component `CImageMaps` handles rendered textures by creating a `Camera` object along them instead of decoding a image file and storing it.
- Implemented a GUI for textures where bound textures and their respective slots are displayed.
  - For rendered textures this view is live and can be used to adjust camera pressing `alt`.
- Reimplemented `TriMesh` class not to rely on `cy::TriMesh`. Now cy::TriMesh is only used for importing `.obj` files.
  - For better shading I implemented re-indexing and duplicating certain vertex attributes as OBJ format uses multiple faced indexing.

Some Screenshots:


File Edit
Control Panel
light
light
Transform Material Mesh Image Maps
Diffuse
From Obj File From rendered img
As Diffuse texture
Remove
Rendered from camera
Camera
Look at Reset Camera
0.000 0.000 0.050 Center
0.000 -64.974 -6.517 Position
0.000 1.000 0.000 Up

File Edit
Control Panel
Scene Objects
plane
teapot
light
light1
Light
Point
0.870 Intensity
R:255 G:255 B:255 Color
250.423 621.332 -199.130 Position
Camera
Look at Reset Camera
0.000 -2.340 0.000 Center
-56.950 14.532 1.006 Position
0.000 1.000 0.000 Up