

CuRLI_v1

CS6610 Interactive Computer Graphics HW repository. Stands for **CompUter Renders Lot of Images**. It is intended 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 convenient.

Building CuRLI for Windows

Building CuRLI for windows requires Cmake 3.0

Dependencies

Most of these dependencies are included as submodules and compile 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

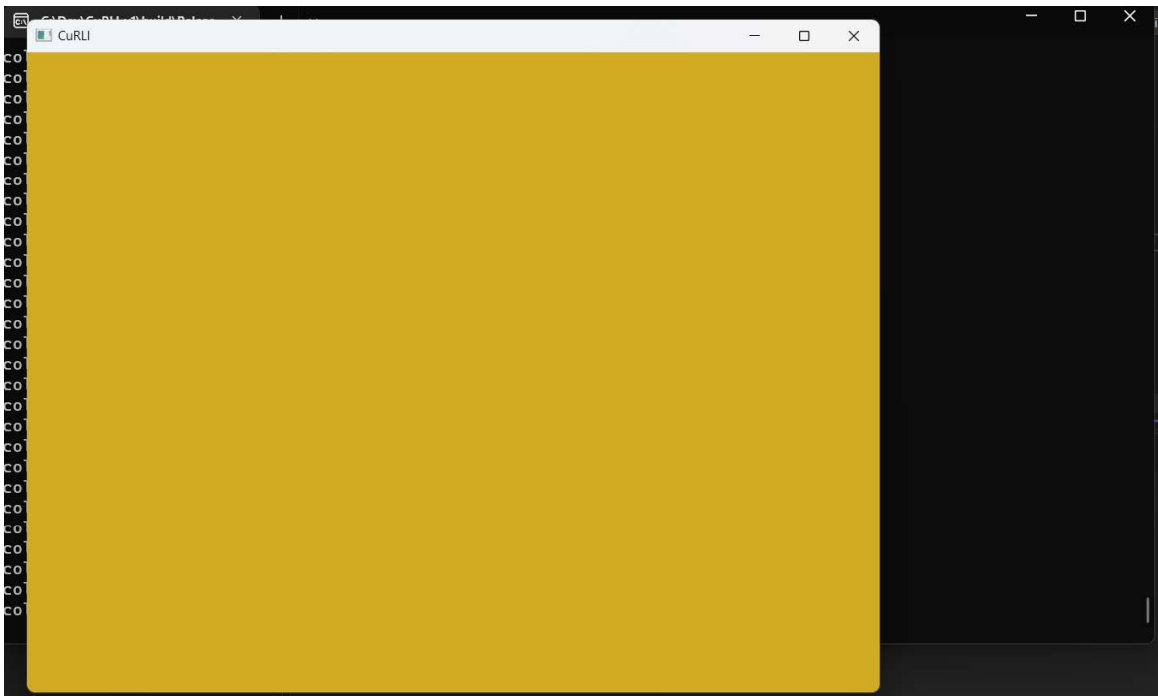
Milestones

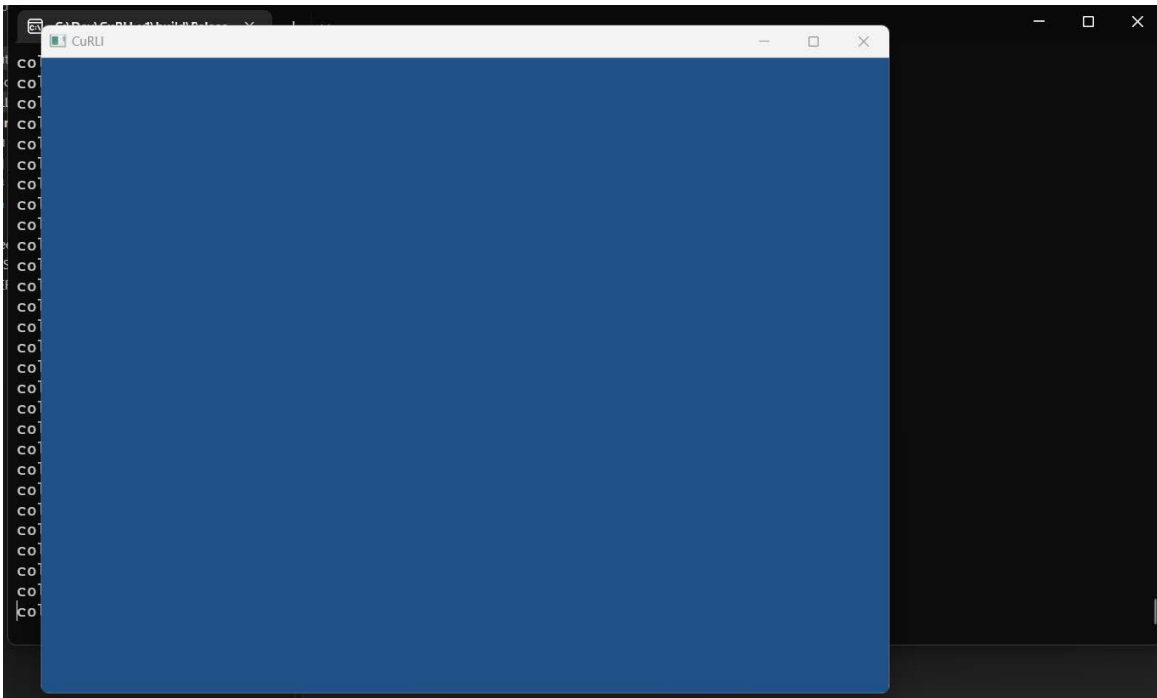
Project 1 - H*llo World

Project 1 requirements:

- ✓ Creating Window context
- ✓ Keyboard listeners where 'esc' is used to call `glutLeaveMainLoop();`
- ✓ Setting window size, position, name and clear color during initialization.
- ✓ 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:

- ✓ 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: `./curl.exe path/to/mesh`
- ✓ Implemented very simple shaders (`/assets/shaders/simple/...`) to transform and render vertex points in a constant single color as `GL_POINTS`.
- ✓ 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*.
- ✓ 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 `curl` launches, pressing `F5` and `F6` will use the edited shaders if compilation is successful.
- ✓ `ImGui` window also includes a button that recenters camera(`F1`) to the mesh center point.
- ✓ Pressing `P` also lets user switch between orthographic and perspective projection types.

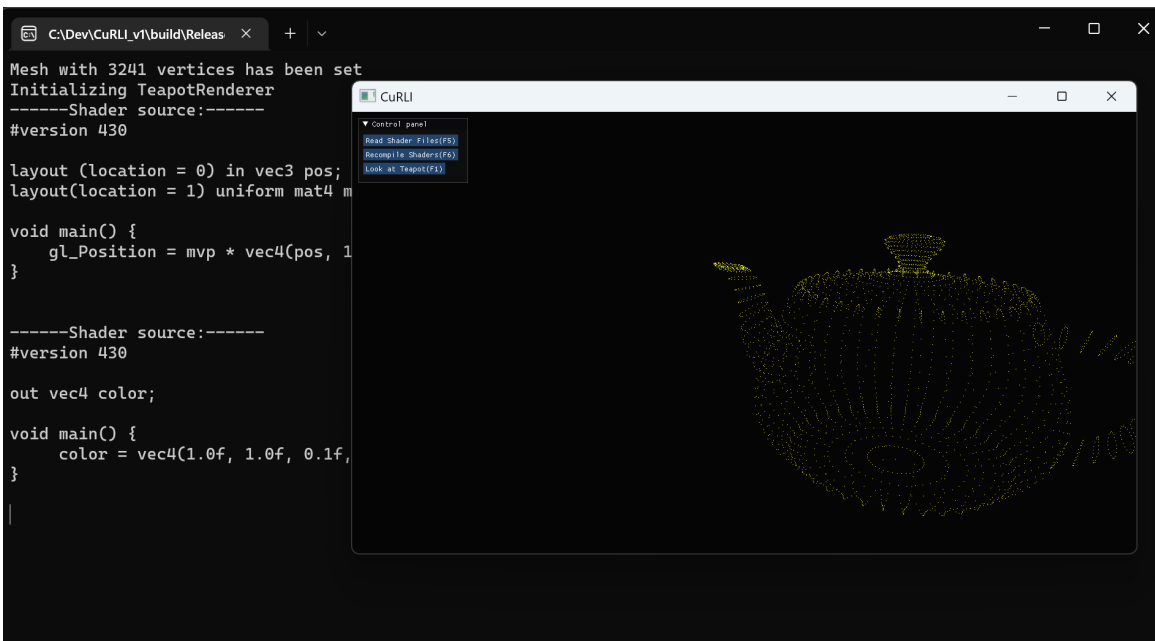
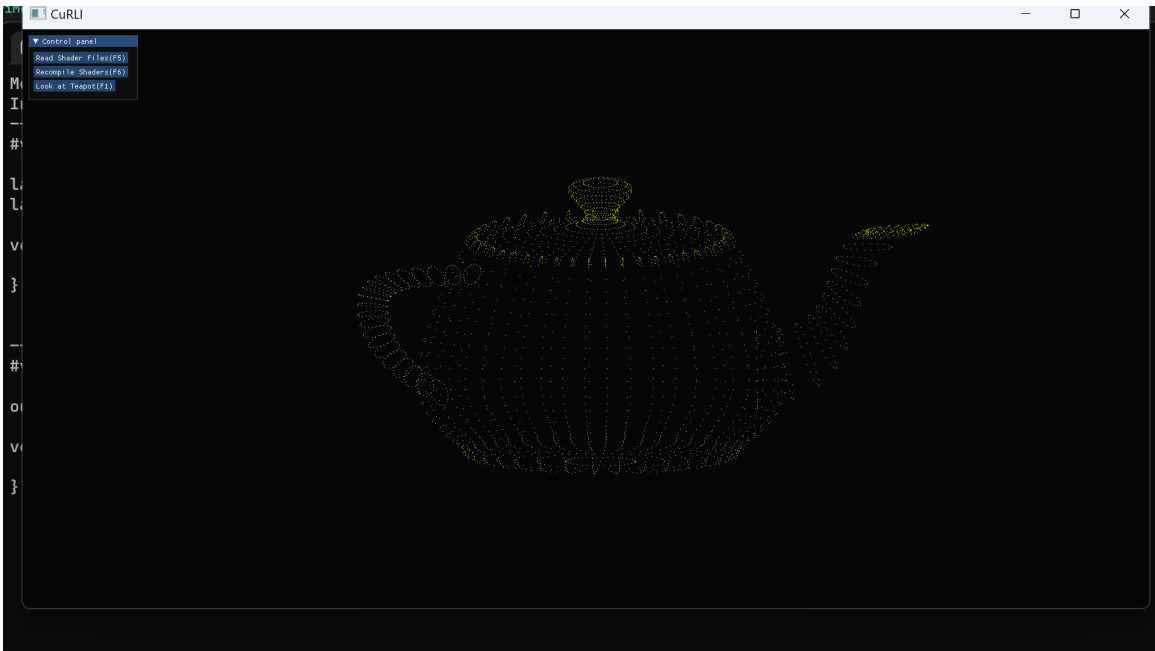
Additional Features:

- [Curiously recurring template pattern](#) has been utilized 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 customizable 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 functions are overridden 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:

- ✓ Displayed triangles instead of points
- ✓ Uploaded and transforming vertex normals using inverse transpose of model view matrix
- ✓ Implemented Blinn-Phong shading in view space using half angles.
- ✓ 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 allows convenience binding shaders and uploading uniforms.
- Using the ECS I let shaders render multiple light sources over multiple objects.

Some Screenshots:

