Department of Computer Engineering
University of Peradeniya
C Project

CO1020 : Computer Systems Programming

June 01, 2025

1 `libtiny3d` : 3D Software Renderer Library

- **Duration:** 4 Weeks
- **Language:** C
- **Type:** Group Project
- **Goal:** Build your own lightweight graphics engine in C from scratch - no OpenGL or DirectX. Just you, math, and pixels.

2 Project Summary

libtiny3d is a 4-week systems programming project where you'll build a complete 3D graphics engine from the ground up using pure C. This isn't just another coding assignment, it's a deep dive into the mathematical and computational foundations that power every 3D application, game, and visualization tool you've ever used.

Core Philosophy: Instead of relying on established graphics libraries like OpenGL or DirectX, you'll implement every component yourself. This means understanding and coding the fundamental algorithms that transform mathematical descriptions of 3D objects into pixels on your screen.

- A complete software rendering pipeline that takes 3D geometric data and produces 2D images.
- Custom mathematical libraries for 3D transformations, vector operations, and matrix calculations.
- A floating-point canvas system that can draw smooth lines and shapes with sub-pixel precision.
- Lighting systems that simulate how light interacts with 3D objects.
- Animation frameworks using advanced interpolation techniques.
- A working demo that showcases animated, lit 3D wireframe objects.

3 Project Tasks

3.1 Task 1: Canvas & Line Drawing

A digital sheet (canvas) where you can draw smooth lines, even at diagonal or floating-point positions like (3.5, 7.7).

3.1.1 Implementation

- `canvas_t` structure
  - **File:** `canvas.h`
  - create a structure called `canvas_t`
  - It should store:
    - The width and height of the canvas.
    - A 2D array of float values between 0.0 and 1.0 representing brightness at each pixel.
- `set_pixel_f(canvas, x, y, intensity)`
  - **File:** `canvas.c`
  - Takes floating-point coordinates (like 2.3, 4.7) and a brightness value.
  - It spreads the brightness across nearby 4 pixels using **bilinear filtering**. (Don't just round to the nearest pixel!)
- `draw_line_f(canvas, x0, y0, x1, y1, thickness)`
  - **File:** `canvas.c`
  - Use DDA algorithm to draw a smooth line from `(x0, y0)` to `(x1, y1)`.
  - Make sure lines support a **thickness** value (draws wider lines).

3.1.2 Demo

In `main.c`, draw lines going out from the center of the canvas at 15° steps, like a clock face.

3.2 Task 2: 3D Math Foundation

Your own math engine to handle 3D vectors and 4×4 matrices, so you can move and rotate 3D objects.

3.2.1 Implementation

- `vec3_t` structure
  - **File:** `math3d.h`
  - Holds a 3D vector using x, y, z (Cartesian), and also r, θ, ϕ (spherical).
  - Automatically update spherical when Cartesian changes and vice versa.
- `vec3_from_spherical(r, theta, phi)`

  - **File:** `math3d.c`
  - Converts spherical coordinates into x, y, z.

- `vec3_normalize_fast()`
  - **File:** `math3d.c`
  - Makes the vector have length 1 using a **fast inverse square root trick**.

- `vec3_slerp(a, b, t)`
  - **File:** `math3d.c`
  - Smoothly moves from direction a to direction b by a fraction t (0 to 1). Used for smooth turning or rotation.

- `mat4_t` struct and operations
  - **File:** `math3d.h` + `math3d.c`
  - Stores a 4×4 matrix (use column-major layout internally).
    * `mat4_identity()`
    * `mat4_translate(tx, ty, tz)`
    * `mat4_scale(sx, sy, sz)`
    * `mat4_rotate_xyz(rx, ry, rz)`
    * `mat4_frustum_asymmetric(...)` to build a different perspective.

3.2.2 Demo

Create a cube in `test_math.c` and manually apply transforms to move and project it.

3.3 Task 3: 3D Rendering Pipeline

A way to take a 3D object, **transform** it, and draw it on your 2D canvas as a **wireframe**.

3.3.1 Implementation

- `project_vertex()`
  - **File:** `renderer.c`
  - Takes a 3D point and applies the full transformation:
    * Local $\longrightarrow$ World $\longrightarrow$ Camera $\longrightarrow$ Projection $\longrightarrow$ Screen.

- `clip_to_circular_viewport(canvas, x, y)`
  - **File:** `renderer.c`
  - Checks if a pixel is inside a circular drawing area (to clip it).

- `render_wireframe()`
  - **File:** `renderer.c`
  - Draws a 3D object as a bunch of lines (wireframe).

  - Uses `draw_line_f()` to draw each edge between two projected vertices.
  - Use the depth (Z) value to sort lines **from back to front**.
    * You can implement a **logarithmic Z-buffer** to do this.

- `generate_soccer_ball()`
  - **File:** `main.c` or helper file
  - Create a **truncated icosahedron** (soccer ball shape) using code or load from file.
  - Connect edges to form the wireframe.

- Bonus: Quaternion Rotation
  - Rotate the object using **quaternions** to avoid rotation bugs (like gimbal lock).
  - Use `slerp()` to smoothly change direction.

3.3.2 Demo

Show the rotating soccer ball wireframe, clipped to a circle.

3.4 Task 4: Lighting & Polish

Make your wireframes look better using **lighting** and smooth **animations**.

3.4.1 Implementation

- Lambert Lighting
  - **File:** `lighting.c`
  - Compute how bright each line (edge) should be:
    * If a line is pointing toward the light, it should be bright. Use: `intensity = max(0, dot(edge_dir, light_dir))`
  - Add support for **multiple lights**.

- Bézier Animation
  - **File:** `animation.c`
  - Use cubic Bézier curves to smoothly move objects over time.
  - Function: `vec3_bezier(p0, p1, p2, p3, t)` returns position at time `t`.
  - Animate multiple objects together, synced.

- Looping & Sync
  - Make sure animations loop back smoothly (start = end).
  - Use the same time step to sync all moving objects.

3.4.2 Demo

Render multiple lit wireframe shapes moving smoothly in sync.

4 Example Project Structure

The filenames and what they contain are suggestions. However, we expect you to maintain the directory structure as is, along with **Makefile**, **README.md**, and the **GroupXX_report.pdf** in the mentioned locations.

```
libtiny3d/
├── include/
│   ├── tiny3d.h
│   ├── canvas.h
│   ├── math3d.h
│   ├── renderer.h
│   └── lighting.h
├── src/
│   ├── canvas.c
│   ├── math3d.c
│   ├── renderer.c
│   └── lighting.c
├── tests/
│   ├── test_math.c
│   ├── test_pipeline.c
│   └── visual_tests/
├── demo/
│   └── main.c
├── build/
│   ├── libtiny3d.a
│   └── demo
├── documentation/
│   └── GroupXX_report.pdf
├── README.md
└── Makefile
```

5 Submission

- Complete **libtiny3d** source code as a zip file named **GroupXX_project.zip** while ensuring sufficient comments and best coding practices are used, as this will be considered in marking.
- A working demo to showcase all implemented functionalities. Creativity and innovative implementations will be accounted for during marking.
- A `README.md` file with detailed instructions for building and using the library.
- A PDF report named **GroupXX_project.pdf** which includes all implementation details, mathematical explanations, design decisions, and a section for AI tool usage. Also, make sure to include a section on individual contributions.
- A 10-minute recorded presentation (screenshare + voice) named **GroupXX_project.mp4** explaining the implementations and design decisions, followed by a demo showcasing the usage of the library.

Important Note on AI Tools:
Submitting code generated entirely by AI tools (e.g., ChatGPT, GitHub Copilot, etc.) **without demonstrating proper understanding or design effort**—will result in zero marks. *We evaluate not only the correctness of your code but also your understanding and ability to design and implement a solution.* Please ensure that your submission reflects your own learning. **Make sure to mention how, when, and where it was used and how you verified the correctness of its outputs in the report.**

6 Deadline

- The deadline is **(30th June 2025) 24:00h(midnight)**.