Design Decisions Document

**CS-330 Computer Graphic and Visualizations**

**Final Project**

**Development Choices and Scene Design**

The 3D scene recreates a modern developer workstation with stereo speakers, a monitor, keyboard, and mouse. Several key design decisions were made to ensure both visual accuracy and code maintainability.

**Object Selection and Composition**

The scene consists of five main objects chosen for their geometric simplicity while maintaining visual fidelity:

1. **Stereo Speakers:** Implemented using composite shapes including boxes for cabinets, cylinders for drivers, and custom port shapes. The speakers demonstrate the ability to combine multiple primitive shapes into complex objects while maintaining proper proportions.
2. **Monitor:** Created using a combination of:
   1. Box mesh for the screen and frame
   2. Tapered cylinder for the stand
   3. Box for the base

This design captures the essential characteristics while keeping the polygon count low.

1. **Keyboard:** Simplified to two main components:
   1. Base box for the main body
   2. Slightly elevated surface for the keys

This abstraction maintains visual recognition while avoiding the complexity of individual key modeling.

1. **Mouse:** Composed of:
   1. Box mesh for the main body
   2. Cylinder for the scroll wheel

This approach achieves recognizable form with minimal geometric complexity.

**Implementation Strategy**

The objects were implemented using a modular, class-based approach:

class Monitor {

private:

float m\_width, m\_height, m\_depth;

void RenderScreen(), RenderStand(), RenderBase();

public:

void Render(const glm::vec3& position, float rotationY);

};

This design pattern allows each object to:

* Manage its own internal state
* Handle its own rendering logic
* Scale proportionally when dimensions change

**Scene Navigation and Camera Control**

The camera system was implemented with careful consideration for user interaction:

**Movement Controls**

* WASD keys control horizontal and forward/backward movement
* QE keys handle vertical movement
* Mouse controls orientation with pitch and yaw
* Mouse scroll adjusts movement speed

The camera implementation uses a position-target system:

Camera::ProcessKeyboard(CameraMovement direction, float deltaTime) {

float velocity = MovementSpeed \* deltaTime;

if (direction == FORWARD)

Position += Front \* velocity;

// Similar handling for other directions

}

**Code Modularity and Organization**

The codebase employs several strategies to maintain modularity and reusability:

**Custom Functions**

1. **Object-Specific Render Methods:**

void RenderScreen() {

// Handle screen-specific transformations and materials

}

These functions encapsulate rendering logic for specific components, making the code more maintainable and easier to modify.

1. **Material and Texture Management**

void SetShaderMaterial(std::string materialTag) {

// Apply material properties to shader

}

Centralizes material handling and allows easy swapping of materials.

1. **UV Coordinate Scaling**

void SetTextureUVScale(float u, float v) {

m\_pShader->setVec2Value("UVscale", glm::vec2(u, v));

}

Provides reusable texture coordinate scaling across different objects.

**Scene Organization**

The scene manager employs a hierarchical structure:

SceneManager {

private:

Speaker\* m\_pLeftSpeaker;

Speaker\* m\_pRightSpeaker;

Monitor\* m\_pMonitor;

Keyboard\* m\_pKeyboard;

Mouse\* m\_pMouse;

}

This organization allows for:

* Independent object manipulation
* Simplified scene composition
* Easy addition of new objects
* Clear ownership and cleanup handling

The modular design proved particularly valuable when adjusting object positions and scales, as each object could be modified independently without affecting others in the scene. The final positioning used specific coordinates:

// Monitor centered slightly back

glm::vec3 monitorPosition = glm::vec3(0.0f, 1.5f, -0.5f);

// Keyboard positioned forward of monitor

glm::vec3 keyboardPosition = glm::vec3(0.0f, (-1.0f + 0.15f), 1.0f);

// Mouse to the right of keyboard

glm::vec3 mousePosition = glm::vec3(2.0f, (-1.0f + 0.2f), 1.0f);

These precise positions ensure objects sit properly on the desk surface while maintaining appropriate spatial relationships between components.