COSC363 Assignment

- Max Shi (msh254) (36310991)

I declare that this assignment submission represents my own work (except for allowed material provided in the course), and that ideas or extracts from other sources are properly acknowledged in the report. I have not allowed anyone to copy my work with the intention of passing it off as their own work.

Name: Max Shi

Student ID: 36310991

Date: 31/03/2025

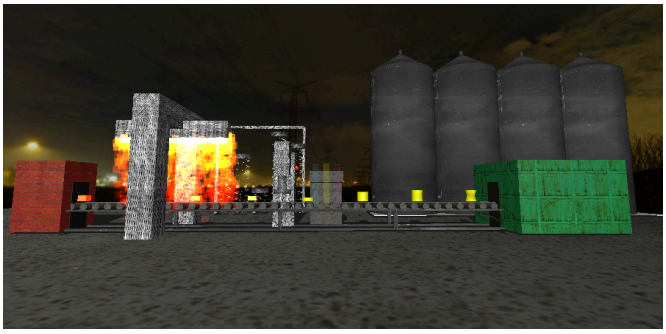
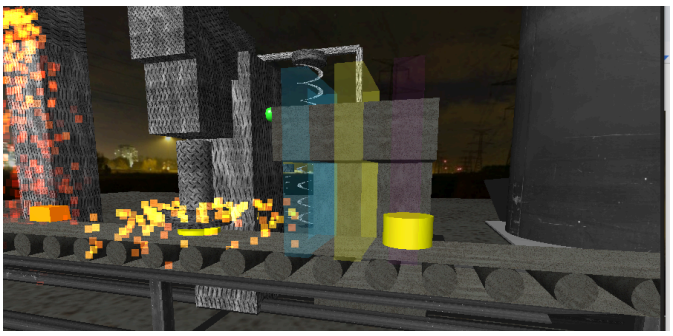
Technical Information

This project was developed using CLion on a Linux Machine (not CSSE lab machine). All credits and licensing is found in the project README or the references section of the report

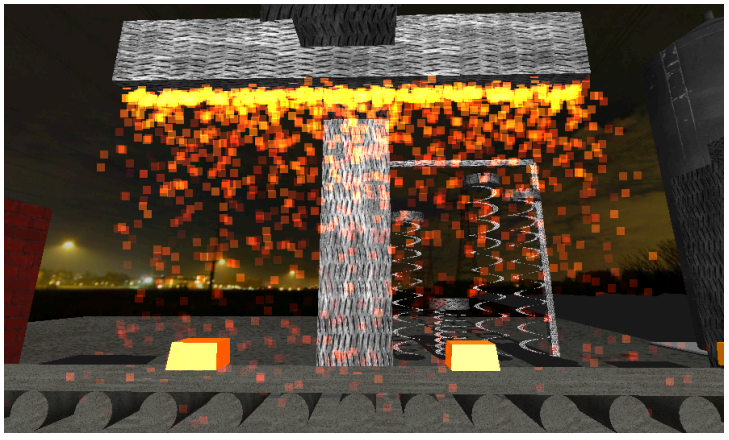
Scene Description

The scene depicts an industrial assembly line, operating in a 'atmospheric' urban environment (in a city). The main feature of the scene is the ingot processing line - an assembly line which transforms ingots into twisted vases. In more detail - the scene consists of a detailed conveyor belt, fashioned with rollers, a detailed base, depicted with stationary and moving parts. The conveyor belt has several mechanical devices, which act to process the item going across the belt. Such devices include fire blasters, a presser, and an upgrader. On each side of the conveyor belt, a kiln produces the hot ingots (at the start of the conveyor belt), and packager is the end point of the belt. In the background, there are several factory structures, which help elevate the industrial 'aesthetic' of the scene. These include oscillating springs, and several metal silos.

Images

- (Above, Left) an overview of the scene, which includes all items in the scene description
- (above, Right) - the presser in operation, producing sparks. Additionally, on the right is the upgrader, which is a series of oscillating 'beams'



- (above) the fire blaster, raw ingots (with emissive lighting), and conveyor rollers
- Note: This image is taken while clipped into one support of the fire blaster

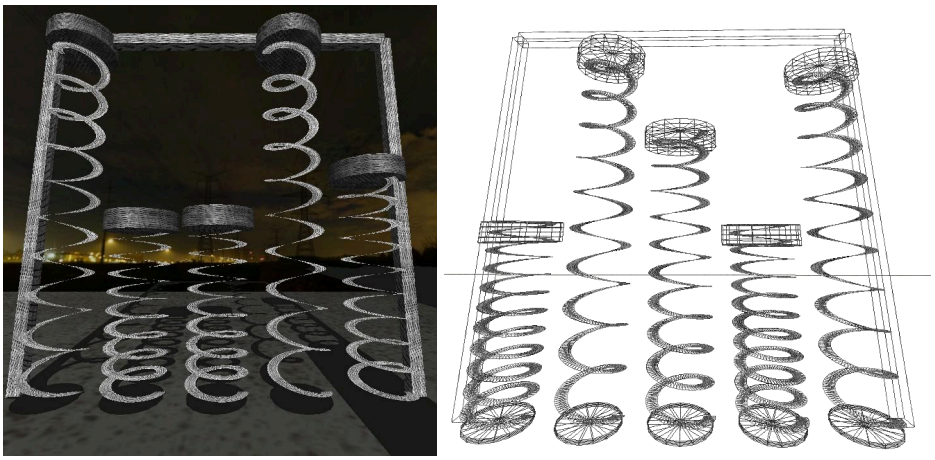
List of Added Features

1. Planar shadows cast by every object in the scene
2. Moving conveyor (animation by texture), and moving rollers

3. Sky Box
4. Custom-Built sweep surface
5. Shape generated by mathematical formula
6. Dynamic mesh
7. Particle system
8. Dynamic time function
9. Extended movement

Diagrams and Sketches for Designing Models

Spring



- (above) images of the springs rendered in the program, one in normal view, one in wireframe mode

The spring in this diagram is generated using both custom-built sweep surfaces, and mathematical surface generation.
The path for the coil is generated by sampling a variate angle in a loop of (0-360 degrees) * SPRING_NUM_COILS
The coordinates for the path are generated as follows: note, $\underline{i}$ is the variate angle in degrees, whereas *angle* is in rads

$$x = SPRING\ RADIUS \times cos(angle)$$
$$z = SPRING\ RADIUS \times sin(angle)$$
$$y = SPRING\ BASE\ OFFSET \times \frac{i}{360 \times SPRING\ NUM\ COILS} \times currentHeight$$

$$currentHeight = (minHeight + (maxHeight - minHeight) \times (0.5 + 0.5 \times sin(beltOffset \times 3.0 + timeOffset))) \times 2$$

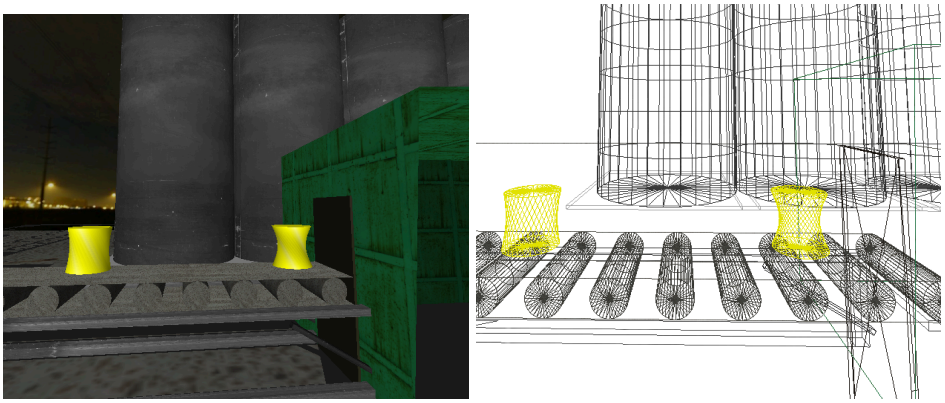
The use of this equation should be quite self-explanatory, the x and z coordinates for the spiral are dependent on the angle, which loops back every 360 degrees, forming a circular shape. This can be seen in code, where;

```
for (int i = 0; i <= 360 * (int)SPRING_NUM_COILS; i += 5) {
    float angle = i * M_PI / 180.0f; // This changes degrees to radians
    float y = SPRING_BASE_OFFSET + (i / (360.0f * SPRING_NUM_COILS)) * currentHeight;
    float xC = SPRING_RADIUS * cos(angle);
    float zC = SPRING_RADIUS * sin(angle);
    float normalX = xC / SPRING_RADIUS;
    float normalZ = zC / SPRING_RADIUS;
    glTexCoord2f((float)i / (360.0f * SPRING_NUM_COILS), 0.0f);
    glVertex3f(xC - normalX * SPRING_THICKNESS, y, zC - normalZ * SPRING_THICKNESS);
    glTexCoord2f((float)i / (360.0f * SPRING_NUM_COILS), 1.0f);
    glVertex3f(xC + normalX * SPRING_THICKNESS, y, zC + normalZ * SPRING_THICKNESS);
}
```

Texture coordinates are assigned along the ***length*** of the spring, calculated via the x and z coordinates.
Note: the height is variable on beltOffset, which is essentially the progression of the belt, which is dependent on time.

Dynamic Mesh - Processed Item (ingot)

A dynamic mesh (in principle) is a model whose geometry changes over time which cannot be achieved solely by applying uniform transformations. This project has a dynamic mesh via a **twist deformation** on the **processed item** in the scene.



- (top, left) the natural photo showcasing the processed item, (top, right) the wireframe mode

Twist Deformation

The twist effect is achieved only when the X position of the cylinder (processed item) has reached a certain threshold. The twist effect only occurs after the 'disk' has increased in height to a cylinder (x 0 to x = 9). Once the cylinder has reached x = 9, then the twisting occurs

Twist amount

The twist amount is determined by the world coordinate (of the item), by the following formula

$$twistFactor = \frac{worldX - 9}{16 - 9}$$ (twistFactor is clamped to a maximum of 1)

and then the twist is applied to the object via

$$twist = twistFactor \times ITEM\_MAX\_TWIST$$

Twist Construction

The side surface of the processed item is constructed by iterating through item segments. For each segment, two vertices are defined: Bottom Vertex and Top Vertex, whose coordinates are determined through

$$x_{bottom} = ITEM\_RADIUS \times cos(\theta),\ z_{bottom} = ITEM\_RADIUS \times sin(\theta)$$

$$\theta top = \theta + twist \times \pi/180$$

$$x_{top} = ITEM\_RADIUS \times cos(\theta_{top}),\ z_{top} = ITEM\_RADIUS \times sin(\theta_{top})$$

When these two vertices are connected via the quad strip, the side surface of the cylinder is deformed, such that the top face is twisted relative to the bottom face. In particular, this deformation is calculated dynamically, meaning each vertex is updated as a function of worldX, which is important as it gives the item the appearance of 'twisting' continuously.

Particle System

The particle system for this project is a dynamic effect which enhances the realism and quality of the scene. In this project, particles are stored in a global array, with the particle struct holding key information pertaining to the particle, such as position, velocity, colour, lifetime, scale and others. As there are two 'machines' which create particles, there are two distinct particle types in this project -being sparks (caused by presser), and fire (caused by fire blaster), which have distinct properties and uses.

When a new particle is created, through functions like *createParticle* or *createFireParticle*, the parameters for the particles are randomised, to simulate natural behaviours. Such can be expressed through equations such as;

$$angle = \frac{rand()\ \%\ 360 \times \pi}{180.0},\ velocity_x = speed \times cos(angle),\ velocity_z = speed \times sin(angle)$$

When the particles are updated, the particle states are updated accordingly, using a simple physics and collision engine

$$x_{new} = x_{old} + v_x \cdot \Delta t,\ y_{new} = y_{old} + v_y \cdot \Delta t,\ z_{new} = z_{old} + v_z \cdot \Delta t$$

This calculates the new position of the particle, but factors influence particle speed, such as gravity, or bouncing

$$v_y = v_y + SPARK\ GRAVITY \times \Delta t,\ v_y = -v_y \times BOUNCE\ DAMPING$$

Due to limitations on page sizes, this will be the extent of detail in which I will go into about the particle system.

Control Functions

- 'q' | 'Q' – Wireframe Mode
- 'w' | 'W' – Move Forward
- 'a' " 'A' – Move Left
- 's' | 'S' – Move Back
- 'd' | 'D' – Move Right
- 'Left Arrow' – Look Left
- 'Up Arrow' – Look Up

- 'Down Arrow' – Look Down
- 'Right Arrow' – Look Right
- '+' | '=' – Increase conveyor belt speed
- '-' | '_'– Decrease conveyor belt speed

Build Instructions

This project can be compiled using the CSSE lab machines via **VSCodium**.

- In VSCode install the CMake and CMake Tools extension
- Create a directory and move all project files including CMakeLists.txt into the directory
- Open the directory in VSCodium
- In the CMake tab, go to Project Status -> Configure, and press Select A Kit button (pencil icon)
- Select linux gcc as the compiler, doing so should create necessary files in a build directory
- Go to Project Outline, and set the build and launch targets if necessary
- Go to Project Status -> Launch, and press the run button (play icon). The project should then compile and run the executable

Compiling using CLion

- Open the project in CLion
- Clion will automatically build the CMakeList file, and configure everything.
- Press the green icon in the top right of the screen 'Run' on main.out

Note: By default, the CMake extension will compile and run the program from the build folder, meaning the paths to texture files will be of format "../model.tga".

Compiling via terminal

- Open the folder containing the project
- Open the terminal (in the folder)
- type 'mkdir build' and 'cd build'
- type 'cmake ..' and 'cmake --build .'
- run the program via './main.out'

References

Helix : https://en.wikipedia.org/wiki/Helix#Mathematical_description
Textures (excluding skybox) were found on PolyHaven, whose assets are all licensed CCO (Creative Commons).
The skybox was found via Humus (Emil Persson), whose assets are all licensed CC BY 3.0 (Creative Commons).
Specifically;
Skybox : https://www.humus.name/index.php?page=Textures&start=131 (first one on the page – no direct link)
Concrete: https://polyhaven.com/a/gravel_concrete_03
Metal: https://polyhaven.com/a/metal_plate
Metal Sheet: https://polyhaven.com/a/blue_metal_plate
Metal Wall:https://polyhaven.com/a/rusty_metal_grid
Brick: https://polyhaven.com/a/red_brick

AI Use

- Models used: ChatGPT 4o, ChatGPT o1, Claude Sonnet 3.7
  - Claude was used to explain topics surrounding shadows, this includes the 'shadow pass' feature, as well as how to compute a shadow matrix
  - ChatGPT 4o and ChatGPT o1 were used to generate docstrings