# CS 330 Final Project Reflection

# Justify development choices for your 3D scene.

# The backdrop utilizes a large, flat plane scaled to `(20.0f, 1.0f, 10.0f)` to create a wide, textured surface resembling sand. The use of a sand texture with a UV scale of `(5.0, 5.0)` allows for tiling, ensuring a detailed appearance without distortion. This choice reflects a beach environment, which is fitting for the underwater setting of the 3D world. The shader material adds realism by simulating light reflection, enhancing the scene's depth.

# For Squidward's house, a tapered cylinder scaled to `(3.5f, 8.0f, 3.5f)` represents the iconic structure. The design choices, including a slanted roof created by adjusting the Z-axis rotation, enhance the whimsical and exaggerated style characteristic of the cartoon. Textures like “blue rock” contribute to the visual authenticity of the environment, while the modular setup for transformations simplifies code management, allowing for easy adjustments.

# Patrick's house is constructed using a combination of shapes: a half sphere for the main body and a lot of cylinders for the antanae. The main structure is scaled to `(5.0f, 3.0f, 5.0f)` to give it a substantial presence. The brown rock texture enhances recognition of Patrick's rock, while rotating the cylinders created a realistic addition to the structure. This combination effectively captures the charm of the animated series while keeping the code organized for future enhancements.

# SpongeBob's house is depicted with a three cylinder structure, scaled to `(4.0f, 5.0f, 4.0f)`, reflecting the character's playful nature. The use of a "glass" texture for the windows allows light interactions that make the scene feel lively. By implementing a modular design for transformations, the code remains efficient and straightforward, facilitating potential updates while maintaining the integrity of the character’s representation.

# The jellyfish is rendered as a small sphere with unique scaling, creating diversity and enhancing the dynamic quality of the scene. The use of a transparent shader allows for light diffusion, mimicking the ethereal look of jellyfish. This design choice aligns with the cartoon's aesthetic, bringing life to the underwater environment. The modular transformation function ensures the jellyfish can be positioned easily throughout the scene, adding to the overall vibrancy.

# Explain how a user can navigate your 3D scene.

# In the 3D scene, users can navigate the virtual camera using both mouse and keyboard inputs, enabling a dynamic exploration of the environment. The mouse facilitates camera orientation adjustments, allowing users to look around the scene effortlessly. Mouse movements trigger the `Mouse\_Position\_Callback` function, which calculates the difference between the current and previous positions to update the camera's yaw and pitch. Additionally, the mouse scroll wheel zooms in and out by adjusting the camera's field of view (FOV) through the `Mouse\_Scroll\_Wheel\_Callback` function.

# Keyboard controls provide straightforward navigation within the 3D space. The `W`, `A`, `S`, and `D` keys allow movement forward, backward, left, and right, respectively. This movement is managed by the `ProcessKeyboardEvents` function, ensuring smooth transitions as the user traverses the environment. The `Q` and `E` keys enable vertical movement, allowing the camera to ascend or descend.

# Moreover, users can toggle between orthographic and perspective projections using the `O` and `P` keys. Pressing `O` switches to an orthographic view, flattening the scene into a 2D-like projection, while `P` reverts to a perspective view that enhances depth perception. This toggling functionality provides flexibility in how the scene is viewed, accommodating various user preferences.

# Explain the custom functions in your program that you are using to make your code more modular and organized.

# The `SetTransformations` function is fundamental, as it sets the scale, rotation, and position of each mesh before rendering. By isolating this transformation logic, it can be reused across multiple objects, ensuring consistent application without redundant code.

# The `SetShaderTexture` function assigns specific textures to objects, allowing different objects to share the same texture while being flexibly applied. This simplifies the rendering of multiple objects with varying textures and makes it easier to add new textures as needed.

# To control texture mapping, the `SetTextureUVScale` function adjusts the UV scale for tiling or stretching textures across an object's surface. This function is reusable whenever a specific UV scale is required, ensuring that textures appear correctly.

# The `SetShaderMaterial` function applies materials to objects, defining properties like diffuse color and shininess. By separating this logic, materials can be easily reused and modified across different objects.

# Lastly, functions such as `DrawBoxMesh`, `DrawCylinderMesh`, and `DrawTorusMesh` encapsulate the drawing logic for each geometric shape. This modular approach allows for any mesh type to be reused throughout the scene without rewriting code.

# Overall, these custom functions streamline the rendering process, maintain clean and organized code, and ensure scalability when adding or modifying objects within the 3D scene.