1. Pop-Up World

Part A:

1. Develop a 3D scene that will be similar to a platform game level. Add objects, paths, etc., and use various textures to decorate your scene.
2. Add an avatar that the player would be able to control in the 3D mode. The avatar could be human-like, an animal, or an object.
3. Implement basic physics for 3D objects and avatar (e.g., gravity, collision detection).
4. Implement a camera system (third-person view) that can switch between 2D and 3D views and create a smooth transition effect between 2D and 3D modes. Implement user controls for navigating both the 2D and 3D worlds.

Part B:

5. Implement basic lighting and shadows based on any of the common techniques learned in the lab.
6. Create a mission for the avatar in 3D mode that the avatar would not be able to do in 2D mode (e.g. the avatar should pass through a "secret" path not visible in 2D mode to reach a certain goal [a]). Similarly, create a mission for the avatar that the player should switch in 2D mode to fulfill (e.g. two platforms are connected that are not connected in 3D mode, they seem connected in 2D). Ensure that character movements are consistent during the 2D to 3D transitions.
7. Create a 2D scene and find a way to make it seem cartoon-like.
8. Transition smoothly the camera (first-person view) to reveal the "back" of the scene that would be 3D. Make this camera transition seem like the player is "peaking" in the back of the 2D scene.

Bonus:

1. Create avatar animations that work seamlessly in both 2D and 3D modes.
2. Add special effects (e.g., particle systems) to enhance visual appeal in both modes.

[a]
https://static.wikia.nocookie.net/ultimatepopculture/images/4/46/Super_Paper_Mario_Gameplay.png/revision/latest?cb=20220805130123

2. Lava Lamp

Part A:

1. Load a model of a table into the scene. You may find one on the internet, create your own using modeling software (e.g., Blender), or make it up in code using triangles.
2. Load a lava lamp model into the scene. You may find one on the internet, create your own using modeling software (e.g., Blender), or make it up in code using triangles. If the model includes a glass window or blobs, remove those.
3. Create a directional light emulating the sun using Phong shading. Assign each model an appropriate material.
4. Implement shadows using any technique you want.

Part B:

5. Create a few metaballs moving around inside the lava lamp with various radii. This may be done in the CPU or in shader code.
6. Create a translucent tinted glass window for the lava lamp.
7. Give each metaball a different color using color gradients or colormaps.
8. Use particles to create small air bubbles that rise from the metaballs.

Bonus:

Turn each metaball into a point light to make the effect more realistic. Load more furniture and an entire room with walls around the table and lamp to better showcase the effect.

3. Mountain Hiking

Part A:

1. Randomly generate high altitude terrain in order to imitate a mountainous area.
2. Texture the ground with mountainous terrain. Apply random rotation/flipping to the textures to eliminate repetitiveness. Also add sparse vegetation like plants or trees to the ground.
3. Add a first-person camera mimicking a human character (can walk but not fly).
4. Add a directional light to imitate the sun.

Part B:

5. Add a skybox around the scene.
6. Reduce the movement speed of the character as you move up, due to strong winds. Add sound effects of wind howling depending on how high you've hiked.
7. Add view bobbing to make the slowing down more prominent.
8. Add a fog effect near the peaks of the terrain.
9. Add flying particles to simulate dust and debris.
10. Simulate rotational world distortion the user has to escape from.
11. Simulate scene destruction, by voxelizing and quantizing the scene and objects.

Bonus:

1. Sway the vegetation using the vertex shader. Make the swaying more intense depending on the altitude.
2. Add screen space motion blur in the direction of the wind.

4. Classic 8-ball pool

Part A:

1. Create or load a pool table along with 15 pool balls. Attach appropriate textures so that they are as close to the actual game as possible. Set the camera to allow free movement around the table.
2. Implement lighting and shadow algorithms. Place 10 lights evenly spaced along the rim of the table. Give the user the option to toggle each light by pressing any of the digit keys (0-9).
3. Set the table and balls at the starting position of the game. On a user event, give the cue ball velocity.

Part B:

4. Implement collision detection between the balls and the table. Add friction between the balls and the table.
5. By clicking, the camera is set above the table looking downwards. With this view allow the user to aim the cue, and choose how much force to use for the throw.
6. Implement spin. Modify the previous view to allow the user to determine the spin, by targeting specific parts of the cue ball (through a 2d projection of the cue ball).

Bonus:

Make the game turn based. Implement the rules of the game.

Resources:
https://www.robinwood.com/Catalog/FreeStuff/Textures/TexturePages/BallMaps.html