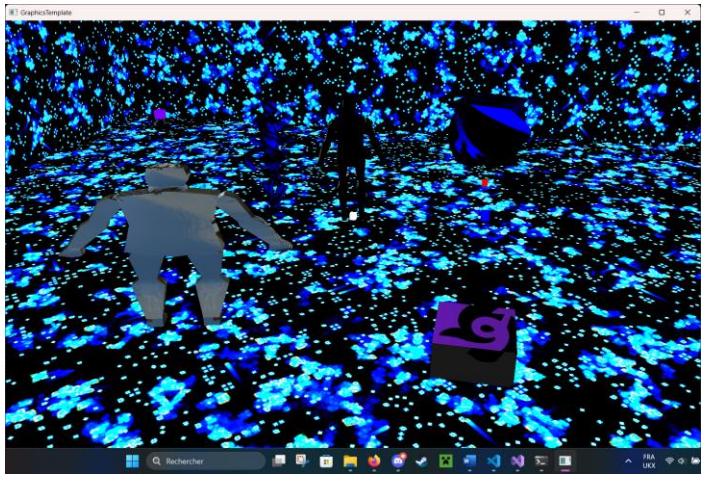
Assessment Report: Creating Virtual Worlds using the GPU 2023-2024

Buisson Felix
Cardiff Metropolitan University,
School of Technology,
Llandaff Campus, Western Ave, Cardiff,
CF5 2YB, Wales, UK
st20278540@cardiffmet.ac.uk

Abstract
In this example document, we describe the different tasks to accomplish to complete this assessment. We will also explain how each task was accomplished, the reasons for the technique(s) selected for their implementation in the engine, as well as the analysis of their suitability within the engine.



Author Keywords
3D assets; Texture; Conway's Game of Life; Animated textures

Introduction

The basic code of the “Engine” was provided to us in advance. The goal of this assessment is to integrate several game elements into the "Engine": import a number of fully textured 3D game assets, attach a camera object to a player character to create a third person setup, write our own shaders and add infrastructure to the engine, here we have chosen animated textures. All of these features are assembled in a single working environment. In this document, we are focusing on creating a "Conway's Game of Life" effect for an anime texture, using OpenGL, GLFW and compute shaders. Conway's Game of Life is a cellular automaton invented by British mathematician John Horton Conway. It plays on an infinite two-dimensional grid, where each cell can be either alive or dead. Cells evolve according to a set of simple rules: 1) A living cell with 2 or 3 living neighbors remains alive. 2) A dead cell with exactly 3 living neighbors becomes alive. 3) A living cell with less than 2 living neighbors dies. 4) A living cell with more than 3 living neighbors dies. 5) The game starts with an initial configuration of living and dead cells, then evolves with each "generation".

Background & Research

The different stages of the Compute Shader pipeline:

Per-vertex processing: In this step, each vertex of the geometric primitives is processed individually. Transforming model coordinates to view space and applying projection matrices to obtain screen coordinates.

Perspective transform & clipping: Once the vertex coordinates are transformed, perspective is applied to give a depth effect. A process is done to not display parts that are outside the camera view.

Rasterization: Primitives are transformed and divided into individual fragments based on their position on the screen.

Per-pixel processing: Each fragment is then processed individually to determine its final color. This includes applying textures, shaders, lighting, etc.

Framebuffer: The processed fragments are combined to form the final image that will be displayed on the screen.

Modern GPUs are more efficient and allow shared resources to have random write capability. APIs allowing you to write algorithms for the GPU without the difficulties associated with using different types of algorithms have been implemented:

Different APIs usable with OpenGL:

OpenCL is a cross-platform compute API. It uses the OpenGL extension architecture to add new features.

OpenGL Compute is a new addition to OpenGL. This integrates compute capability into the OpenGL rendering pipeline and resolves OpenGL/OpenCL interaction issues.

Implementation

GLFW for window management: GLFW offers a simple interface for creating windows and managing user input, making it an ideal choice for our virtual world. The window will be used to display the virtual world, 3d models, animated textures generated by the compute shaders.

```
glfwInit();
GLFWwindow* window = glfwCreateWindow(width, height, "My Virtual World", nullptr, nullptr); // Create a window
glfwMakeContextCurrent(window);
```

We then load the calculation shader.

```
GLuint computeShader = loadComputeShader("life.comp"); // Loading the computing shader
```

We create data texture.

```
while (!glfwWindowShouldClose(window)) {
    glUseProgram(computeShader);
    glBindImageTexture(0, cellTexture, 0, GL_FALSE, 0, GL_READ_WRITE, GL_RGBA8);
    glDispatchCompute(numGroupsX, numGroupsY, 1);
    glMemoryBarrier(GL_SHADER_IMAGE_ACCESS_BARRIER_BIT);

    renderTexture(cellTexture); // Animated texture display

    glfwSwapBuffers(window); // Window update
    glfwPollEvents();
}
```

Compute Shaders for Conway's Game of Life: Compute shaders provide efficient parallelization of calculations on the GPU, which is crucial for simulating large numbers of cells in the game of life. Using compute shaders, we can calculate cell states in a parallel manner, which improves performance and allows smooth animations even with a large amount of cells. It was provided to us during our lab session, here is what it looks like:

```
#version 460

layout (local_size_x = 16, local_size_y = 16) in;

layout (rgba32f, binding = 0) uniform image2D dataIn;
layout (rgba32f, binding = 1) uniform image2D dataOut;

void main() {

    vec4 oldSim = imageLoad(dataIn,ivec2(gl_GlobalInvocationID.x,gl_GlobalInvocationID.y));
    int neighbours = 0;
    int me = ( oldSim.r > 0.5f)?1:0;
    int newMe;

    for(int i = -1;i<=1;i++)
        for(int j=-1;j<=1;j++)
            if(!(i==0&&j==0))
                if(imageLoad(dataIn,ivec2(gl_GlobalInvocationID.x+i,gl_GlobalInvocationID.y+j)).r > 0.5f)
                    neighbours++;

    if(me==1)
        newMe = (neighbours==2||neighbours==3)?1:0;
    else
        newMe = (neighbours==3)?1:0;

    vec4 result = vec4(newMe, 0.9*(oldSim.g+oldSim.r), 0.9*(oldSim.g+oldSim.b), 1.0);

    imageStore(dataOut, ivec2(gl_GlobalInvocationID.xy), result);
}
```

Integration into a Game Engine: The implementation of the animated texture will be integrated into the rendering loop of the game engine. At each iteration of the rendering loop, the compute shaders will be called to update the states of the cells and update them. day animated texture. The animated texture can then be used as a normal texture in the renderer to display visual effects in the virtual world.

```
// Loop while program is not terminated.
while (!glfwWindowShouldClose(window)) {

    game->Update(window);           //update game logic
    scene->Update(window);          //update renderables based on update Game
    scene->Render();                // Render into the current buffer
    glfwSwapBuffers(window);        // Displays what was just rendered (using double buffering).

    glfwPollEvents();               // Use this version when animating as fast as possible
}
```

Rendering Graphics: The generated animated texture will be used as a texture to display dynamic visual effects in the virtual world. It can be applied to objects in the virtual world to simulate special effects.

Evaluation

To improve the implementation of the "Conway's Game of Life" effect, here are some ideas:

- Optimization of Compute Shaders: the computing shader is quite basic in the code provided. For optimal performance, one should optimize the shader to reduce the number of reads and writes to the texture, use techniques such as shared memory to improve parallelization efficiency.

- Memory Management: Effective memory management is crucial to avoid performance loss.

- User Interface and Game Settings: For a better user experience, an intuitive user interface to change life game settings in real time would be nice.

As part of a larger game project:

- Dynamic Visual Effects: The game of life effect could be integrated to create dynamic visual effects in the game world. For example, it could be used to simulate vegetation growth, creature migration patterns, or even weather phenomena.

- Game Mechanics: the Game of Life could be integrated as a game mechanic in its own right. For example, in a real-time strategy game, cells could represent resources or units, and the rules of the game of life could be used to simulate the growth and evolution of colonies.

- Dynamic Narration: the game's evolution of life could be used to generate dynamic events in the game world, thereby influencing the narrative and player choices. For example, random events based on the emergent patterns of the game of life could add variety and unpredictability to gameplay.

Games use similar mechanics. For example, Spore uses a cell simulation to represent the evolution of species through different stages of the game.

Rule-based simulations are being studied in areas such as artificial intelligence and complex systems modeling, showing their potential to enrich the gaming experience.

Games like Minecraft have demonstrated the appeal of procedurally generated worlds and simulation systems to players, presenting interest in dynamic and evolving elements in games.

Conclusions

The implementation of the effect of "Conway's Game of Life" made it possible to discover several important aspects:

A thorough understanding of the rules and emergent behaviors of "Conway's Game of Life" is necessary to make an interesting simulation.

The use of Compute Shaders which offer powerful computing capacity, making them ideal for simulations.

GLFW provides a simple interface for creating windows and managing user input.

Optimizing compute shaders and memory management is crucial to achieving optimal performance, especially when simulating large grids of cells.

The game of life effect can be used in a variety of ways to enrich the gaming experience, ranging from dynamic visual effects to being integrated as a full-fledged game mechanic.

The game of life effect should be integrated into the engine's gameplay loop, where it can interact with other game systems and be updated consistently with the rest of the virtual world.

The implementation of the game of life effect should be designed in a modular and reusable way, so that it can be easily integrated into different game projects without having to rewrite a large part of the code.

Acknowledgements

We thank all the volunteers, and all publications support and staff, who wrote and provided helpful comments on previous versions of this document.

References

John H. Conway. "The Game of Life". Scientific American, 1970.

Gardner, Martin. "Mathematical Games: The fantastic combinations of John Conway's new solitaire game 'life'". Scientific American, 1970.

GLFW official website : glfw.org
Forum GLFW : glfw.org/forum

Akenine-Möller, Tomas, et al. "Real-Time Rendering". CRC Press, 2018.

Rost, Randi J. "OpenGL Shading Language". Addison-Wesley Professional, 2018