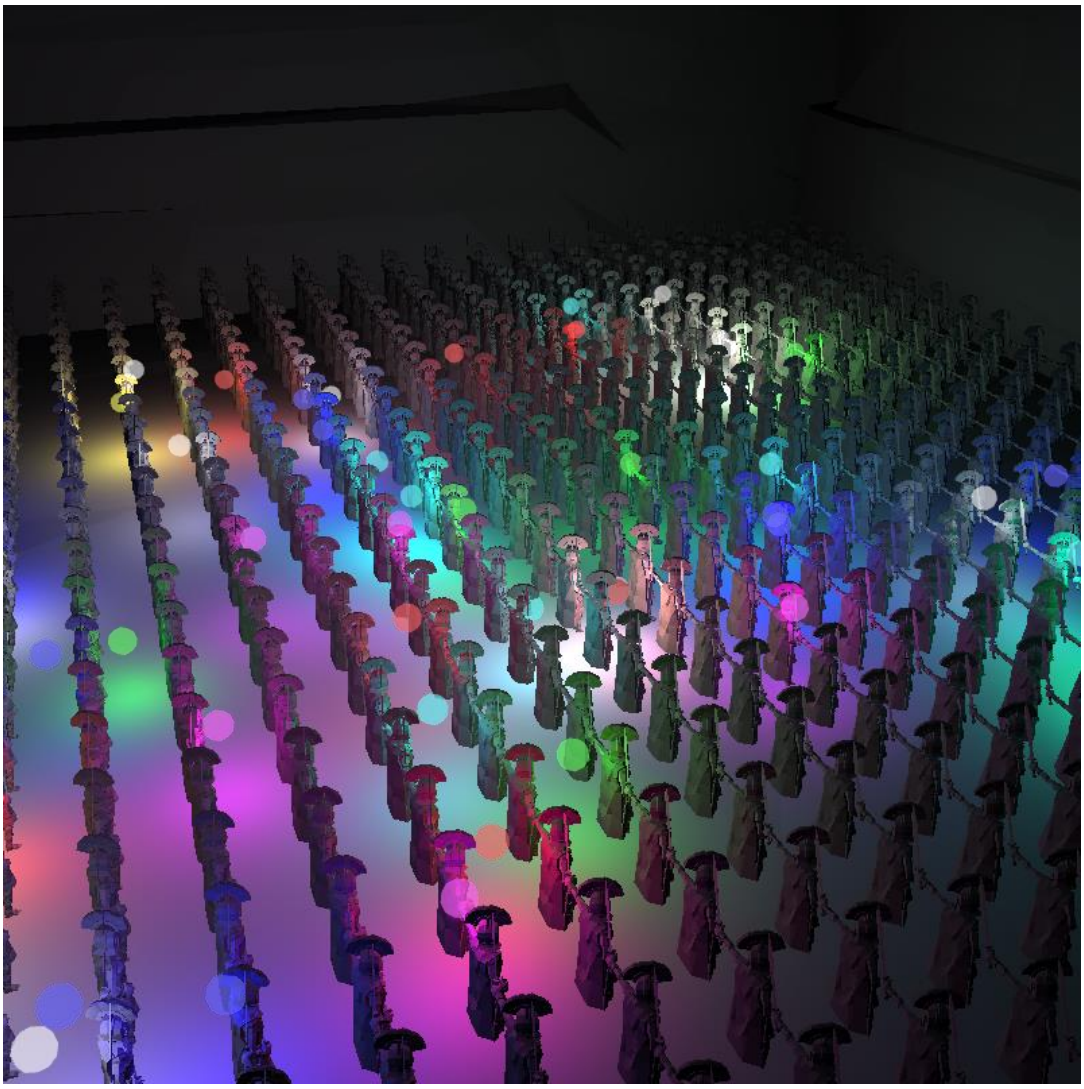
GD2P04 – Advanced Graphics for Games

More Framebuffers and Shaders (40%)



[Example of Deferred Rendering]

Component code and name	GD2P04
Assignment name	More Framebuffers and Shaders
Weighting	40%
Submission deadline	Week 12
Week issued	Week 8

Brief

Instructions/Requirements

Create a technical demo with multiple scenes (input to switch) to showcase the following:

- **Scene 1: Shadows (25%)**
  - The scene is loaded by pressing the key '1'.
  - Create a scene with multiple 3D objects/models, a terrain, and 2 directional lights.
  - Create a shadow map for each directional light using a framebuffer.
  - Apply the shadow map(s) to the scene objects along with standard Blinn-Phong lighting.
    - All objects should be capable of casting their shadow onto any object.
    - Complicated geometry such as terrain should be capable of self-shadowing.
    - Add a shadow bias and Percentage Closer Filtering (PCF) to the shadows to increase quality and remove shadow acne.
  - Designate one of the 3D objects as movable on all 3 global axes and update the shadow maps and cast shadows accordingly as the object moves.

- **Scene 2: Deferred Rendering (25%)**
  - The scene is loaded by pressing the key '2'.
  - Render a scene consisting of at least a flat plane, 10 point lights, and 20 3D models with Blinn-Phong lighting calculations applied.
  - Use the **deferred rendering** pipeline:
    - Geometry pass: Render all geometry in the scene to a G-buffer.
    - Screen-space lighting pass: Render the G-buffer to a screen-space quad using Blinn-Phong lighting on each pixel, calculating the combined effect of all lights in the scene.
  - Represent each point light in the scene as a semi-transparent unlit and untextured (pure color only) 3D object (light source object). Color each light source object to match the color of the light that it represents.
    - Use the **forward rendering** pipeline to add the light source objects to the scene with the correct depth and blending with the deferred rendered scene.

- **Scene 3: Compute Shader (GPU Particles) (25%)**
  - The scene is loaded by pressing the key '3'.
  - Create a **firework** particle system that uses the compute shader and associated buffers to control the movement of each particle.
    - Create 2 parts to the firework particle system:
      - An upwards velocity with a trail effect.
      - A spherical explosion in all directions.
    - Play multiple (at least 4) instances of the firework particle system when the ‘F’ key is pressed. This should be repeatable.
      - Each firework should be a different or randomized color.
      - Randomize each trail time so that the fireworks explode at different times.
      - Once the particle system has run its course, it is disabled (not continuous).
  - Use GL_POINTS for the draw type in the draw call for all particle systems.
  - Ensure blending is enabled with each particle individually fading based on its remaining lifetime percentage (1 -> 0).

- **Scene 4: Tessellation Shaders + Level of Detail (LOD) (15%)**
  - Scene is loaded by pressing the key '2'.
  - Render a triangle using a triangle patch and apply and appropriate texture.
    - Use barycentric coordinates to calculate the tessellated coordinates.
    - Set the inner tessellation levels to 7.
    - Set the outer tessellation levels to 5.
  - Render a quad using triangle patches.
    - Apply Level of Detail (LOD) with the tessellation levels increasing as the camera moves closer and decreasing as the camera moves further away.
  - Render a terrain using tessellation to generate additional points (at least 32x32 sections).
    - Calculate the height value using a passed in heightmap texture in the TES.
    - Apply and appropriate texture to the terrain.

- **Programming Practices (10%)**
  - A ReadMe.txt file is included stating the functionality, controls, and any additional triggers or needed information for the project to showcase all included features.
  - Function headers are consistent and present across all files and functions.
  - Comments are used to clarify the purpose and use of data and functions demonstrating an understanding of the key areas of related code.
  - Classes and functions are appropriately used to create systems and demonstrate a higher understanding of modular code and proper C++ OOP concepts.
  - consistency of naming conventions, code formatting, and accessors to increase readability across all files.
  - No warnings are generated during the build that originate from student project files.
  - No Intermediate files are included.

Guidelines

Follow the guidelines, standards, and specifications regarding the tasks outlined in the instructions section.

The source code is required to display the following features:

- Compiling code:
  - Code must build as submitted in both debug and release.
  - No errors or warnings (originating from student files) at warning level 3 for all build targets.

Submission Guidelines

Place the work in a .zip file and submit it to Blackboard by the time and date specified (see Blackboard).

Naming conventions

The file structure and file names of the submission must follow the file hierarchy listed below.

🗍 YYYY-MM-DD – GD2P04 – Assignment2– *Student Name*.zip
  🗁 Source – *Student Name*
    🗋 Assignment 2.sln
    … Project and source code, etc.
  🗍 Release Build – *Student Name*.zip.
    … Include additional resources and DLLs

Submission structure

Source code folder (Source runs in Visual Studio 2019):

- Solution file (.sln).
- Project file (.vcproj).
- Source files (.cpp, .h).

Release build zip:

- Standalone executable (.exe).
- Any additional files required to run the executable.
- Readme file (.txt).

Intermediate and repository files are removed to reduce file size.

Submission Policy

See the component overview for details.