

Vulkan Cloth Simulation

...

Aidan Ream and Claire Ogawa

Our Goals

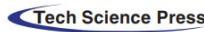
- Vulkan: the good, the bad, and the ugly
- Understand and implement Compute Shaders

Mass-Spring Cloth Simulation Using Compute Shaders

Computer Systems Science & Engineering

DOI:10.32604/csse.2022.020685

Article



Parallel Cloth Simulation Using OpenGL Shading Language

Hongly Va¹, Min-Hyung Choi² and Min Hong^{3,*}

¹Department of Software Convergence, Soonchunhyang University, Asan, 31538, Korea

²Department of Computer Science and Engineering, University of Colorado Denver, Denver, CO 80217, USA

³Department of Computer Software Engineering, Soonchunhyang University, Asan, 31538, Korea

*Corresponding Author: Min Hong. Email: mhong@sch.ac.kr

Received: 03 June 2021; Accepted: 05 July 2021

Abstract: The primary goal of cloth simulation is to express object behavior in a realistic manner and achieve real-time performance by following the fundamental concept of physic. In general, the mass–spring system is applied to real-time cloth simulation with three types of springs. However, hard spring cloth simulation using the mass–spring system requires a small integration time-step in order to use a large stiffness coefficient. Furthermore, to obtain stable behavior, constraint enforcement is used instead of maintenance of the force of each spring. Constraint force computation involves a large sparse linear solving operation. Due to the large computation, we implement a cloth simulation using adaptive constraint activation and deactivation techniques that involve the mass–spring system and constraint enforcement method to prevent excessive elongation of cloth. At the same time, when the length of the spring is stretched or compressed over a defined

We use this paper as inspiration for creating a cloth simulation in Vulkan.

Why Vulkan?

- Vulkan offers finer grain control over the GPU than OpenGL does
- This comes with (if done well) performance optimizations
- Pre-compiled shaders using SPIR-V

Create Structs

```
VkImageView createImageView(VkImage image, VkFormat format, VkImageAspectFlags aspectFlags) {
    VkImageViewCreateInfo viewInfo{};
    viewInfo.sType = VK_STRUCTURE_TYPE_IMAGE_VIEW_CREATE_INFO;
    viewInfo.image = image;
    viewInfo.viewType = VK_IMAGE_VIEW_TYPE_2D;
    viewInfo.format = format;
    viewInfo.subresourceRange.aspectMask = aspectFlags;
    //viewInfo.subresourceRange.aspectMask = VK_IMAGE_ASPECT_COLOR_BIT;
    viewInfo.subresourceRange.baseMipLevel = 0;
    viewInfo.subresourceRange.levelCount = 1;
    viewInfo.subresourceRange.baseArrayLayer = 0;
    viewInfo.subresourceRange.layerCount = 1;

    VkImageView imageView;
    if (vkCreateImageView(device, &viewInfo, nullptr, &imageView) != VK_SUCCESS) {
        throw std::runtime_error("failed to create image view!");
    }

    return imageView;
}
```

(int)0

Search Online

Dependencies

- VulkanSDK
- GLM
- GLFW
- stb_image.h
- TinyOBJLoader <3

Initial Setup

1. Instantiate vulkan (connection between app and the Vulkan library)
 - a. Extensions for loading additional functionalities and validation layers for error checking
2. Use GLFW to construct window
3. Check the hardware to pick graphics card

Hello Triangle Setup

1. Set up queues to interface with hardware
(logical device)
2. Create the swap chain
 - Vulkan has no default framebuffer
3. Create image view structure to store images
(represent attachments)
4. Setup color and depth attachments (define
format, behavior)

The screenshot shows a GitHub code editor interface for a repository named 'vulkanClothSim'. The file 'main.cpp' is open, containing C++ code for a Vulkan application. The code includes various #include directives for GLFW, GLEW, glm, and other libraries, along with defines for GLFW_INCLUDE_VULKAN and GL_KHR_NO_RESETTING. A commit message 'Red Triangle. On the Screen. Yeah Baby.' is visible above the code area.

```
#define GLFW_INCLUDE_VULKAN
#include <GLFW/glfw3.h>
#define GLK_FORCE_RADIANS
#define GLK_PRAGMA_DISABLE_DEPRECATION_WARNINGS
#include "Debugging.hpp"
#include <glm/vec4.hpp>
#include <glm/wavetek.hpp>
#include <optional>
#include <vector>
#include <exception>
#include <cstdint>
#include <cstdlib>
#include <vector>
#include <optional>
#include <utility>
#include <limits> // necessary for uint32_t
#include <limits> // necessary for numeric_limits
#include <algorithm> // necessary for std::clamp
#include <fstream> // necessary for file management
```

Hello Triangle Setup continued

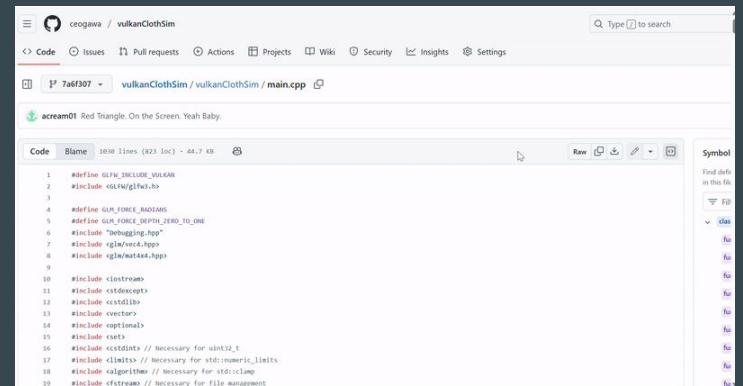
5. Load in shader modules, configure rasterizer and color blending then store in graphics pipeline struct

6. Create framebuffer that references image views

7. Create command pool to allocate command buffers

8. Create command buffer to send commands to GPU

9. Create semaphores for synchronization

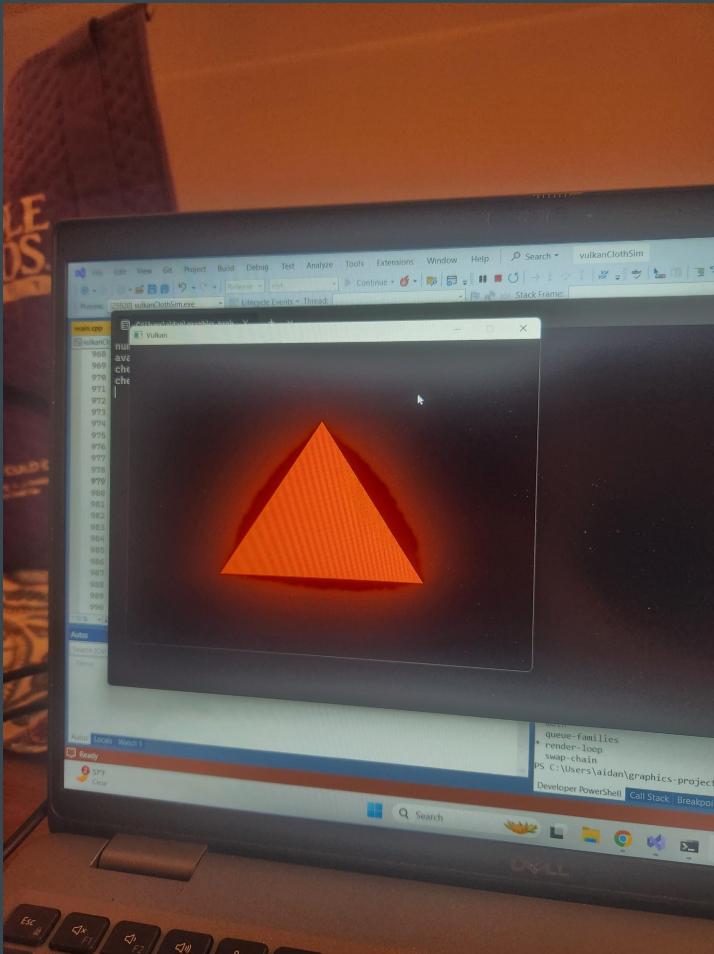


The screenshot shows a GitHub code editor interface for the file `vulkanClothSim/main.cpp`. The code is a C++ program for a Vulkan Cloth Simulation. It includes several header files and defines GLFW_INCLUDE_VULKAN. The code is annotated with comments explaining its purpose, such as "Red Triangle. On the Screen. Yeah Baby." and "Find defn in this file". The GitHub interface includes navigation bars for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings.

```
#define GLFW_INCLUDE_VULKAN
#include <GLFW/glfw3.h>
#define GLFW_FORCE_INIANS
#define VK_NO_PROTOTYPES
#include "Debuging.hpp"
#include <glm/vec4.hpp>
#include <glm/wav4x4.hpp>
#include <vector>
#include <exception>
#include <cstdint>
#include <vulkan/vulkan.hpp>
#include <optional>
#include <set>
#include <stdint.h>
#include <limits> // necessary for uint32_t
#include <algorithm> // necessary for std::clamp
#include <fstream> // necessary for file management
```

Hello Triangle

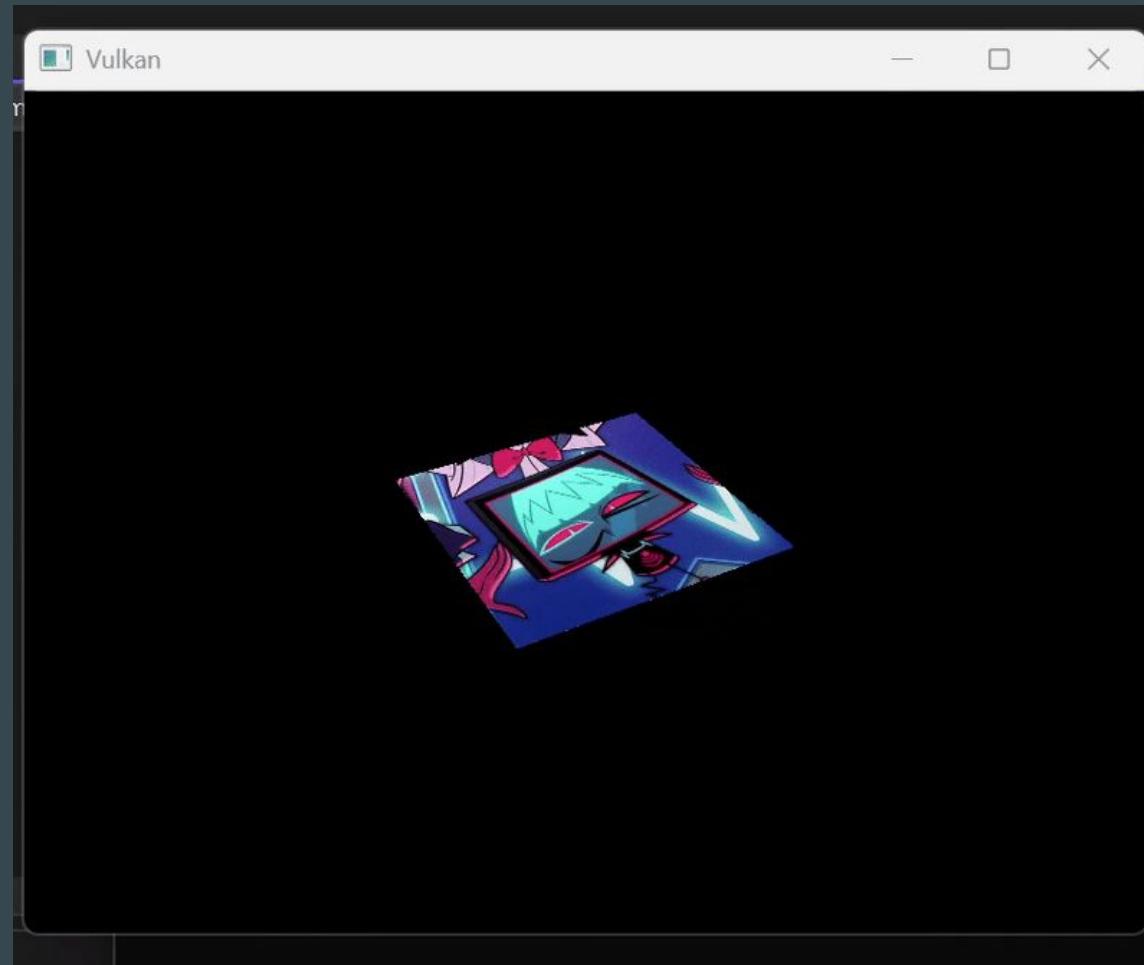
1028 lines of code later, we have a red triangle rendered on the screen!



MVP + Textures

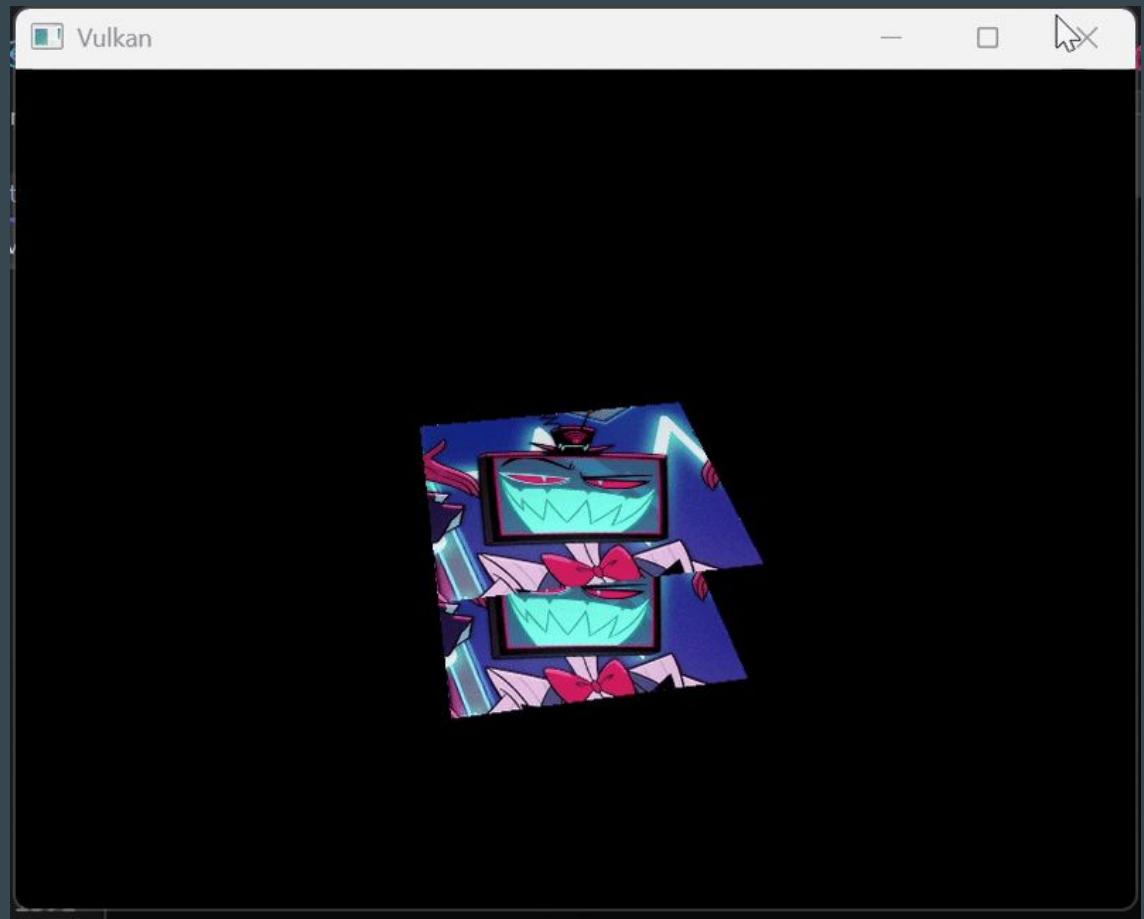
Passed a UBO to the vertex shader with MVP

Loaded in texture image,
created sampler object for
fragment shader

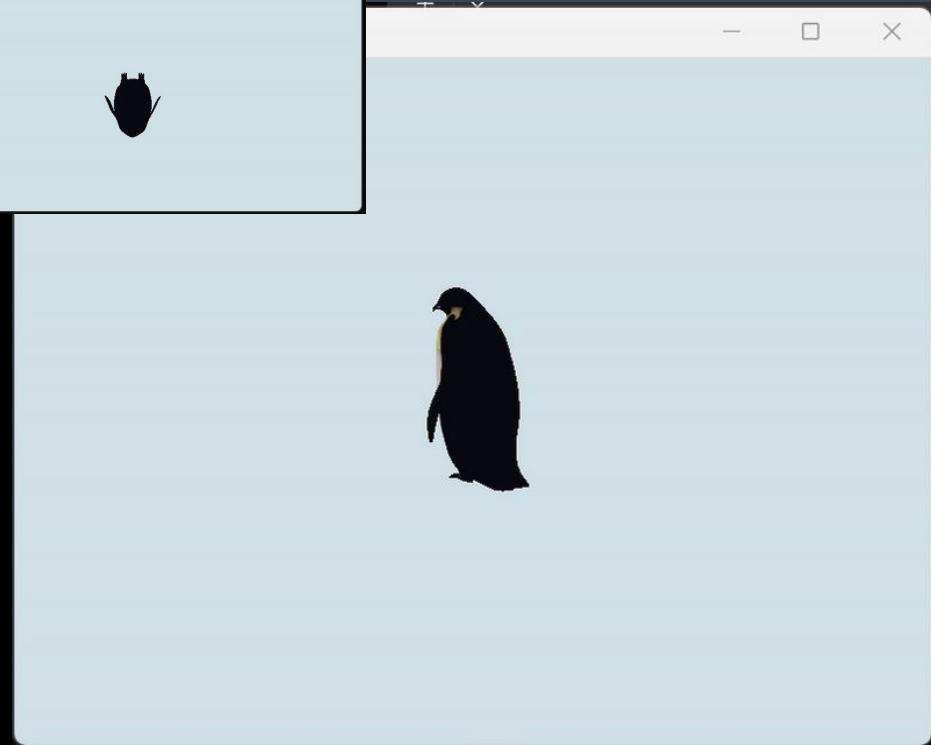
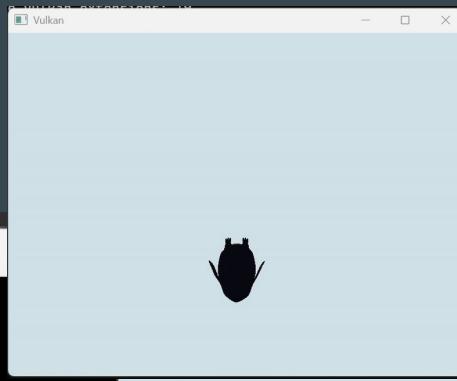
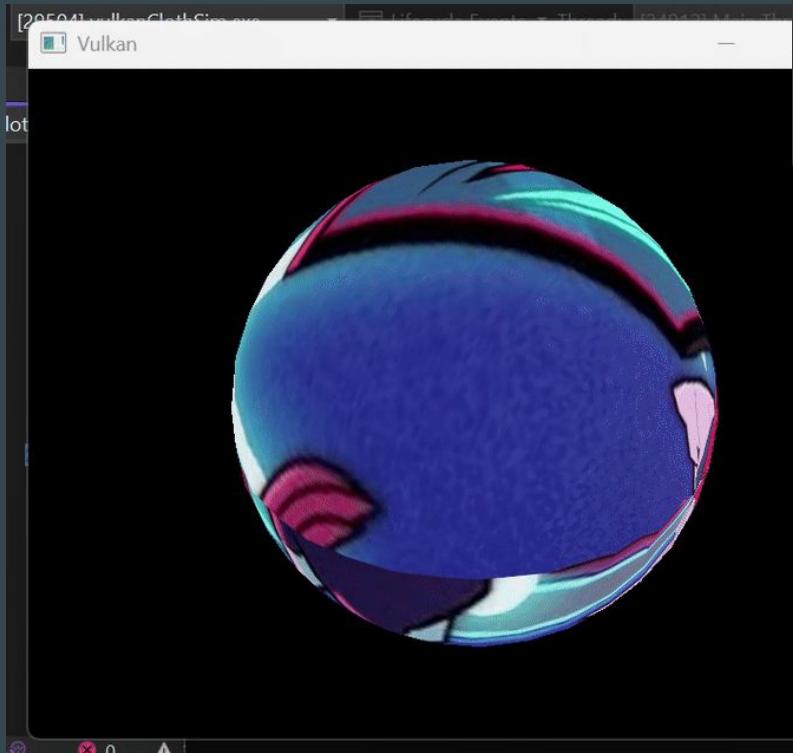


Depth Buffering

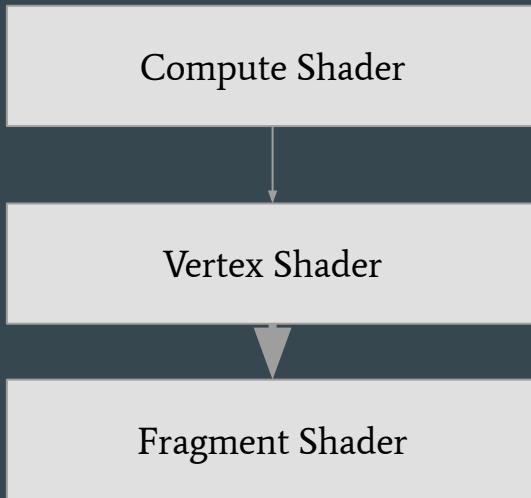
Add a new attachment for depth, create depth image view



OBJ Maximum Potential



Compute Shader



```
compile.bat X cloth.vert cloth.frag cloth.comp M
compile.bat
1 C:\Users\aidan\vulkan\Bin\glslc.exe cloth.vert -o vert.spv
2 C:\Users\aidan\vulkan\Bin\glslc.exe cloth.frag -o frag.spv
3 C:\Users\aidan\vulkan\Bin\glslc.exe cloth.comp -o comp.spv
4 pause
```

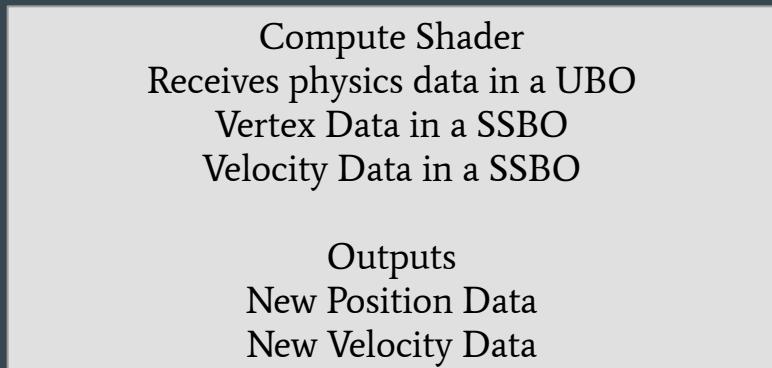


Compute Shader uses SSBOs- Shader Storage Buffer Objects



CPU Defined 50x50
vertex grid

Object Space adjustments based on spring kinematic equations



Normal Graphics Pipeline
Vert -> Frag

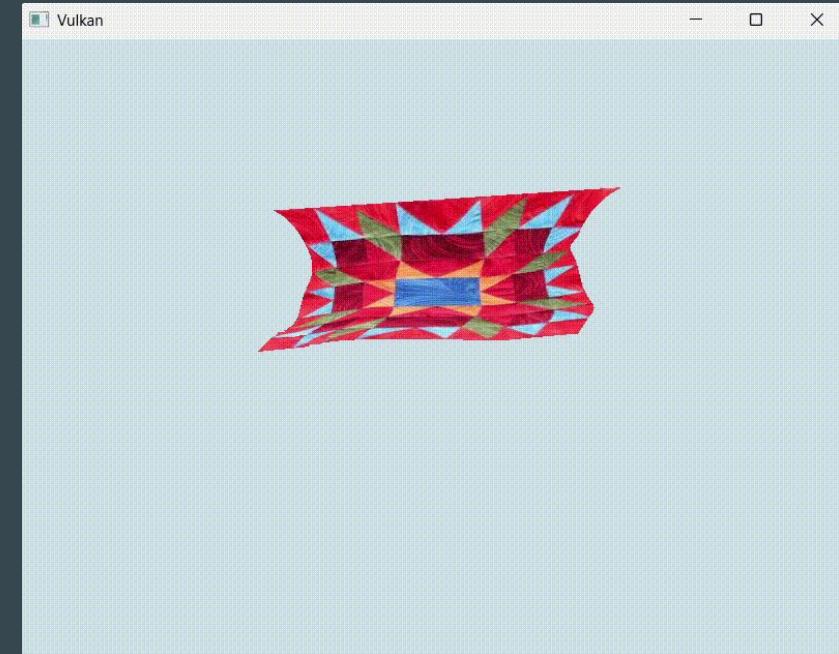
Demo!

Comparing Results

Video Result



Ours

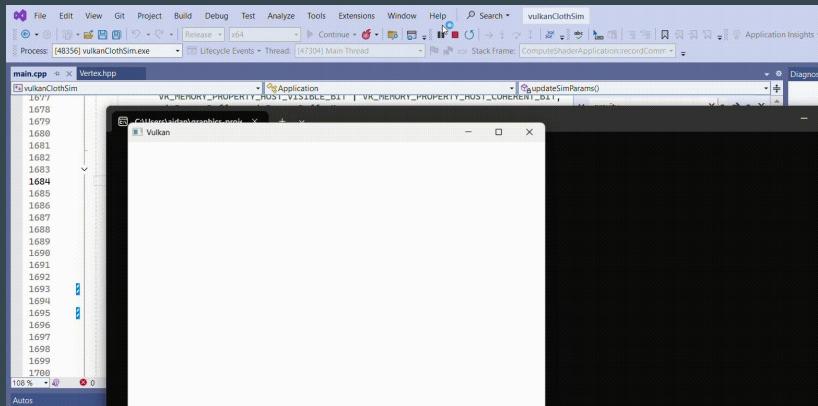


Final Counter

Structs: 72 (not including loops)

Create functions: 27

Main File Length: 2398



THANK YOU

1. Cloth Simulation in Vulkan with Compute Shaders
<https://www.youtube.com/watch?v=D-P-TYLjLJY>
2. Cloth Animation Using The Compute Shader <https://www.youtube.com/watch?v=ms010Fz02wo>
3. <https://vulkan-tutorial.com/Introduction>
4. Va, Hongly, Min-Hyung Choi, and Min Hong. "Parallel Cloth Simulation Using OpenGL Shading Language." *Comput. Syst. Eng.* 41.2 (2022): 427-443.
https://graphics.ucdenver.edu/publications/Parallel_cloth_simulation.pdf