Computer Graphics Project Report

This is the project report for Computer Graphics semester project at the University of Nottingham Ningbo China, which requires us to design a 3D scene in OpenGL. In this report, I will illustrate the implementation details for my program. First, I will showcase my hardware/software settings and used OpenGL libraries (Section 1). Then I will present the overview of my program (Section 2). Next, I will describe how I achieve the requirements in the project description by presenting the achievements for each requirement in my 3D scene (Section 3). Finally, I will show the user guide which contains detailed instructions on how to compile/run/use my program.

1 Hardware/Software Settings

Hardware: AMD Ryzen 7 5800H CPU, NVIDIA Geforce RTX 3060 GPU.
Software: Visual Studio 2019, OpenGL Mathematics 0.9.9.8 [1], The OpenGL Extension Wrangler Library (GLEW) 2.1.0 [2], and freeglut [3].

2 3D Scene Overview

This section will introduce the overview of my 3d scene, in terms of the major components in the scene. Figure 2 shows the snapshot of my 3d scene. The theme of my design is an open-field amusement park resided by a river. The major components in the scene are:

1. Amusement Park Facilities: Ferris Wheel, Pirate Ship
2. Buildings & Decorations: Bridge, Gate, Itinerary boat, Fence
3. Environments: Grass, Running Water, Sky
4. Animals: Dog

The detailed implementations for these components and their corresponding animations will be illustrated in the next section.

3 Implementation Breakdown

This section will break down my implementation in terms of the requirements.

3.1 3D Models

All the models are created **by myself**, either in OpenGL programmed with C++ or handcrafted in Blender. I will first show the design of the models implemented in OpenGL, which is the Ferris Wheel model.
Ferris Wheel: Figure 2 shows the Ferris Wheel in the scene. The implementation is achieved in a top-down manner. First the center axis, rotation wheels, cabins for riders and the bars that connect center axis and rotation wheels are draw, using the basic OpenGL components such as Cylinder, Cube, and Triangles. Next, the base for the ferris wheel is drawn which are entirely consisted of cylinders, with different levels of scaling and rotations.

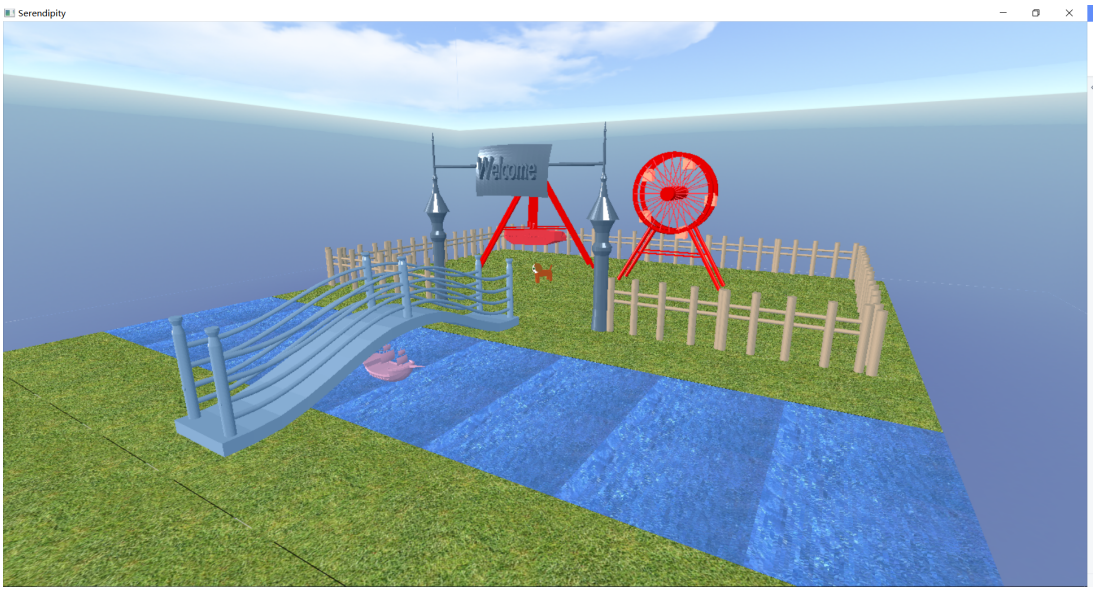

Figure 1: The overview of my 3D scene

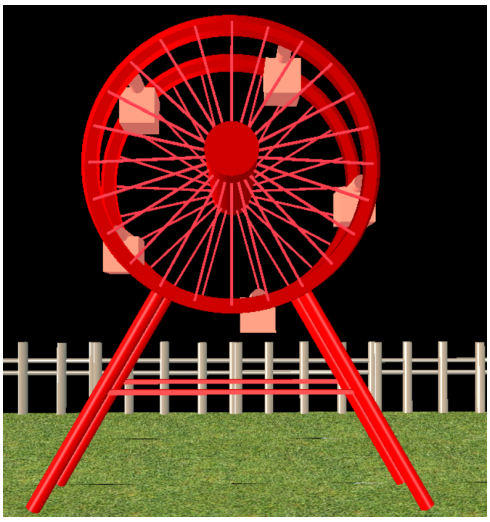

Figure 2: Ferris Wheel

The Dog: Figure 3 shows the dog in the scene. The dog object contains basic OpenGL elements such as sphere, cuboid and cone. The body, legs and the tail is consisted of cuboids (scaled cubes). The head is a sphere which is applied with a dog face texture. The ears of the dog are two tilted cones.

Next, I will present the PirateShip model, which is achieved with the combinations of OpenGL basic components and my self-created model in Blender.

Pirateship: Figure 4 reports the visual appearance of the pirate ship in the scene and Figure 5 shows its appearance in Blender. The boat is created from a cube in Blender. The cube is first scaled into a cuboid. Then the top surface of the cuboid in extruded inside. Then the front and back bottom of the cuboid is scaled to form the curves of a boat. Next, two bars are formed by extruding the surface inside. And the two bars pointing towards sky are also extruded from two regions on the two sides of the boat. Moreover, I attach a Text object to one side of the boat, with words like “Vikings!" to bring about more entertaining atmosphere.

Figure 3: Dog

Figure 4: The pirate ship in the scene

Figure 5: The pirate ship boat in Blender

Finally, I will introduce the models that are completely created in Blender by myself and imported into the scene using *ModelLoader* class[1].

Gate: Figure 6 shows the view of the gate in the scene and Figure 7 shows its appearance in Blender. The gate contains two towers and one flag saying “Welcome" to the tourists. The two towers are created from two cylinders, with various extruding and scaling in the horizontal plane.

Bridge: Figure 8 shows the view of the bridge in the scene and Figure 9 shows its appearance in Blender. The bridge is modeled in three steps. First, the base components, cylinders for the bars on the bridge and the cuboid which is the floor of the bridge are drawn. Second, the horizontal bars and the floor are subdivided to form multiple sub-regions so that later transformation can be smooth. Third, the region at the center of the bridge is selected as the center of the transformation, and I use the built-in smooth tool in Blender to form the curved bridge.

Itinerary Boat: Figure 10 shows the view of the itinerary boat in the scene and Figure 11 shows its appearance in Blender. The itinerary boat is the most complex model in the scene, which is hard to be explained clear with plain words. I followed an online tutorial (see here) to successfully construct it in Blender. It is actually a tiny pirate ship, but it also serves well for an itinerary boat.

Fence: Figure 12 shows the view of the fences in the scene and Figure 13 shows its appearance in Blender. The fence is created using basic cylinders.

[1]The original source is at: https://github.com/WHKnightZ/OpenGL-Load-Model

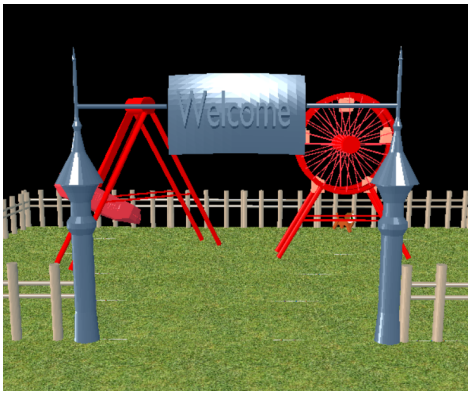

Figure 6: The gate in the scene

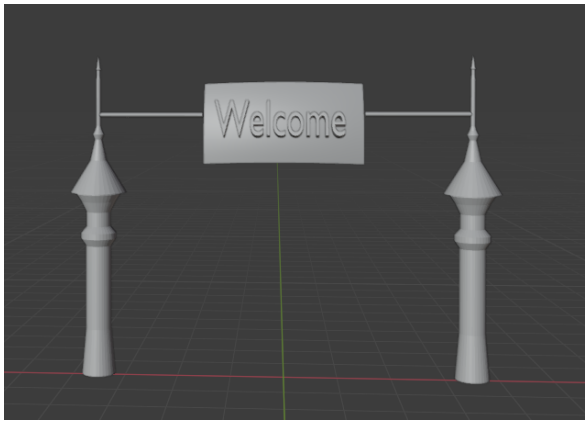

Figure 7: The gate in Blender

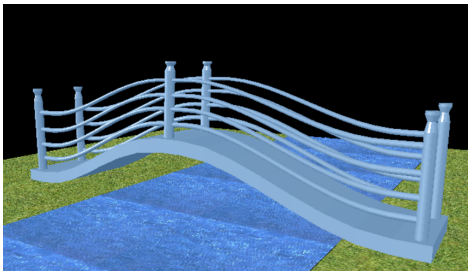

Figure 8: The bridge in the scene

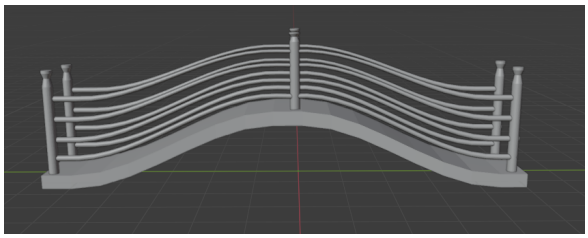

Figure 9: The bridge in Blender

Figure 10: The itinerary boat in the scene

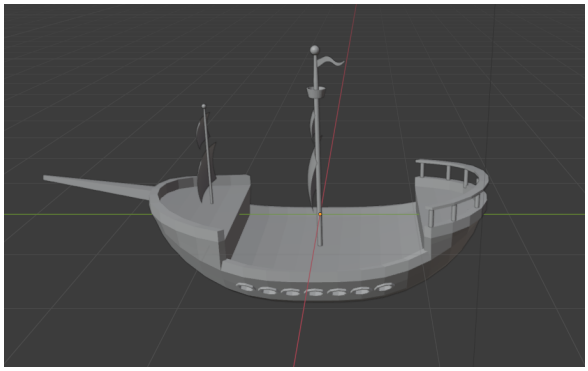

Figure 11: The itinerary boat in Blender

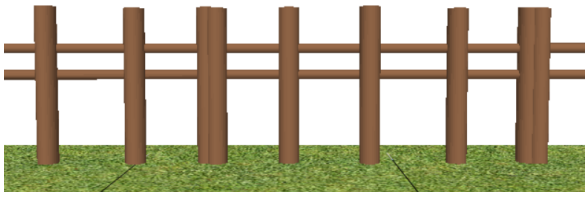

Figure 12: The fences in the scene

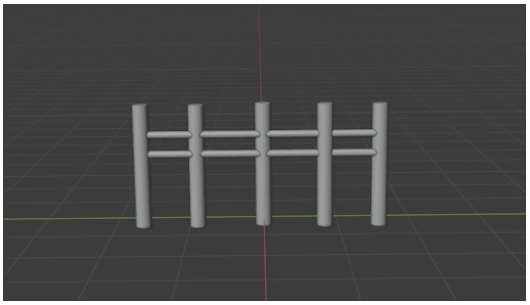

Figure 13: The fence in Blender

3.2 Transformations of the models

I achieve the requirements of transformations of the models by

1. Translating each model to the current layout.
2. Scale several models to fit in. For example, all the models created in Blender has a unit length of 1 for each dimension, the final appearance are all scaled in each dimension.
3. Rotate models for the layouts. For example, the pirate ship model is rotated 40 degrees around the y axis. And the bases for the pirate ship and Ferris Wheel are all rotated cylinders.

3.3 Different Viewpoints

Users can use 'WASD' to move around the scene and drag the mouse to change view directions. In addition, the current camera design enables users to freely fly throughout the scene. This feature is achieved by updating the forward vector of the camera, which represents the forward orthogonal of the camera, to be the same of the vector that contain the values for viewing directions.

3.4 Animations

There are five major animated objects in the scene, which are the Ferris Wheel, the pirate ship, the itinerary boat, the dog and the running water.

For the Ferris Wheel, the animations is the rotation, which its components (i.e., front/back wheels, cabins and bars) are rotated in a timely manner. The rotation speed can be controlled by user inputs. Users can increase the rotation speed by pressing key 'R' and decrease the rotation speed by pressing key 'X'.

For the pirate ship, the animation is a pendulum movement. The max angle for the swing is 60 degrees for each side, and the rotation speed is correlated with the rotated angle compared with the starting points. The mathematical expression for the rotation speed is showed in Equation 1.

$$v = \sqrt{2gl(cos\theta - cos\alpha)} \tag{1}$$

where g is the gravity acceleration coefficient, l is the length of the perpendicular bar on the boat, α is 60 degree and θ is the angle between the bar and the ground.

For the itinerary boat, the animation is a uniform motion following the edges of a Rectangular with width of 500 and height of 60. The current speed is set to 1.5.

For the dog, the animation is wandering around the amusement park in a restricted region. The animation flow for the dog is presented as follows. For each update, the algorithm will first generate random number to decide the direction of the next move. There are total eight options which are upwards (along negative z axis), downwards (along positive z axis), right (along negative x axis), left (along positive x axis), and the combinations of these options (i.e., up-left, up-right, down-left, down-right). Next, the algorithm will check whether the move will step outside of the restricted region, if not, the move will be applied to the dog successfully to achieve wandering animation.

For the running water, the detailed illustrations are shown in Section 3.7. The visual effect of each water block is that the water seems to wave from one side to the other side.

Figure 14: The effects of grass in the scene

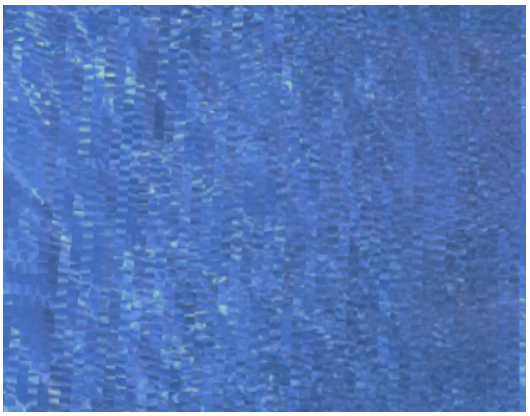

Figure 15: The effects of water in Blender

The brief of the implementation of the running water is that the wave equation and its simplified equation is applied to the vertexes in each water block so that the displacements of the vertexes in each update can control the water texture into different wave forms.

3.5 Textures

Five textures are leveraged in this project, which are for the visual effect of the grass, the running water, the face of the dog, the fur of the dog and the skybox. I use the original texture loader in the G53GRA framework to load the pixels from the bmp files and attach the vertexes of rectangles to show the textures.

Figure 14 shows the visual effects of grass textures in the scene and Figure 15 shows the visual effects of water textures in the scene.

Figure 3 shows the visual effects of the face and fur of the dog. The face texture is applied to an OpenGL sphere and the fur texture is applied to the whole body including the legs, main body and the tail.

Figure 16, 17, 18 and 19 shows the four skybox effects in the scene[2]. The skybox feature is achieved by implementing cubemap. A cubemap contains six textures for each face of the scene. When rendered together, the cubemap constructs a close-to-reality environment in the scene. Users can freely switch different skybox by pressing key 'o' on keyboard.

3.6 Lighting

Two lighting source in different positions are activated in the program, with same configurations in terms of ambience, diffuse, specular. Such design aims for better visual effects as the one single light source cannot lit up all the objects in scene, resulting in extremely dark surfaces for some of the models. In addition, a global model ambient light is activated with a configuration of {0.5, 1.0, 1.0, 1.0} to filter the scene into a cartoon color style.

In addition, the visual effects of the materials of the objects in the scene, in the context of different light source positions and view directions are implemented by invoking the *glMaterialfv()* function. For each objects, the environmental color *GL_AMBIENT*, the diffusing color *GL_DIFFUSE*, the reflection color *GL_SPECULAR* and the shininess

[2]The gate, the bridge, the Pirate Ship and the Ferris Wheel are excluded for better visual inspection

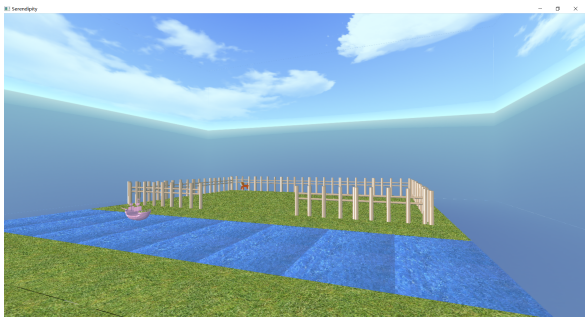

Figure 16: The effects of sunny day skybox in the scene

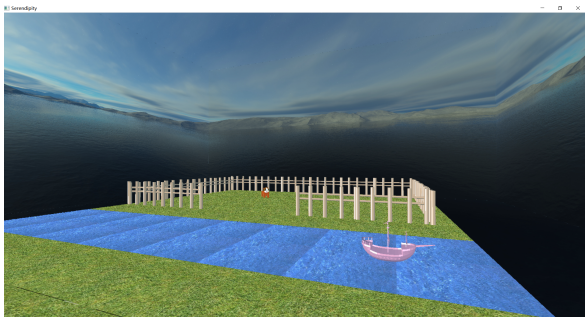

Figure 17: The effects of lake-view skybox in the scene

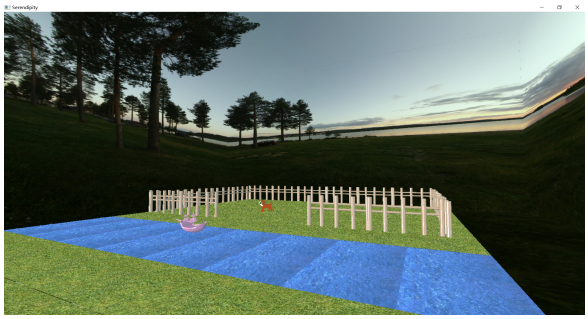

Figure 18: The effects of dawn skybox in the scene

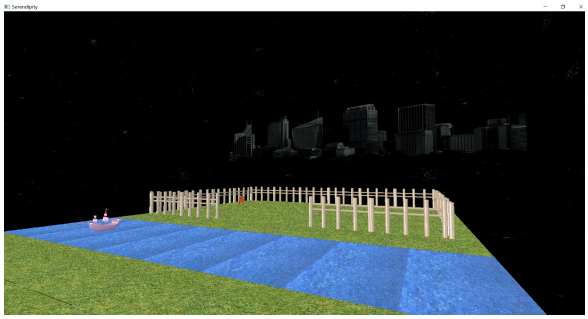

Figure 19: The effects of night skybox in the scene

GL_SHININESS are manually set, so that the objects looks “gleaning" in the light which are more realistic. Figure 21 and Figure 20 show a comparison between the visual effects of before applying and after applying the light effects on the materials of the gate. It is obvious that the gate looks more vivid with light effects for each materials.

3.7 Creative Ideas

In this section, I will present the creative ideas that I think that are unique in this program. As the thought of *creative* can be varied greatly among different users, it would be more fair for uses to personally play around my program to spot anything that are innovative.

First, the ideas of an itinerary boat to patrol around the river is creative. The front scene of my design (i.e., the part that is ahead of the amusement park gate) is quite static without the itinerary boat. The existence of the itinerary boat fuses energy to the scene.

Second, the idea of implementing running water is creative, also considering the fluid simulations are one of the most challenging task in 3D programming. The current algorithm is an implementation of the Wave Equation [4] for the physic equations of the fluid flows. The key equation for the implementation is shown in Equation 2:

$$\frac{\partial^2 z}{\partial t^2} = c^2 \left(\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} \right) - \mu \frac{\partial z}{\partial t} \tag{2}$$

where c the wave speed, μ is the friction coefficient, t is the time and x, y, z is the coordinates.

The above equation can be simplified into the following equation for ease of compu-

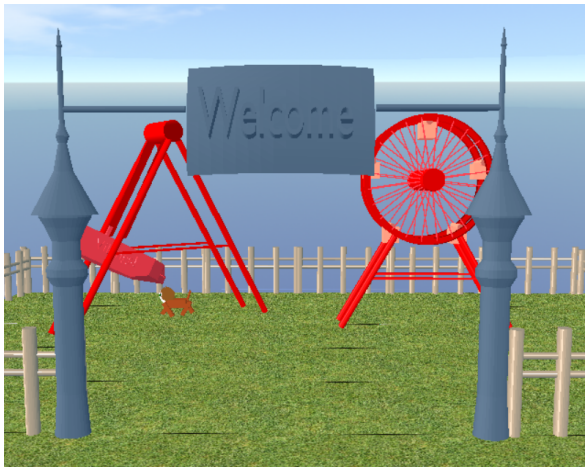

Figure 20: The view of the gate with no light effects for materials in the scene

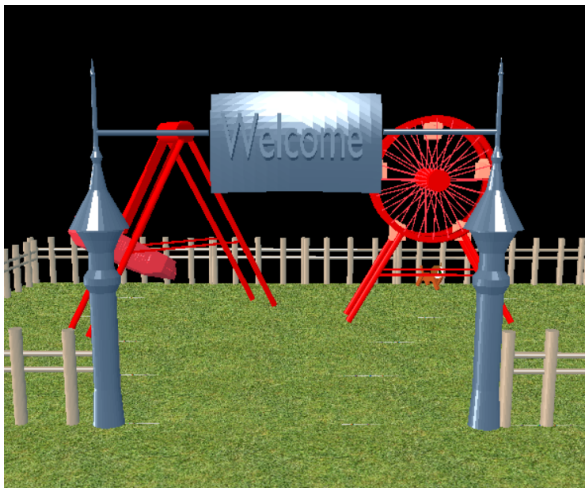

Figure 21: The view of the gate with light effects for materials in the scene

tation:

$$z(i,j,k+1) = \frac{4-8c^2t^2/d^2}{\mu t+2} z(i,j,k) + \frac{\mu t-2}{\mu t+2} z(i,j,k-1) \\ + \frac{2c^2t^2/d^2}{\mu t+2}[z(i+1,j,k) + z(i-1,j,k) + z(i,j+1,k) + z(i,j-1,k)] \tag{3}$$

The next step is to triangulate the surfaces of the floor and assign the coordinates of each triangular to apply loaded water textures[3]. I refer the interested readers to [5, 6] for more details on mathematical explanations for fluid simulations.

Third, the design of a randomly running dog in the amusement park in creative. The path of dog's moves are random, therefore it simulates the movement of a real dog to a great extent.

Fourth, the idea of skybox is creative. Plain single color background of the scene lacks the realistic atmosphere. However, the existence of the sky brings my 3d scene more close to the reality and brings more harmony. Also, users can freely choose the sky effect that is the most attractive to them, to fit in their moods.

Fifth, the idea of implementing light effects of the materials of each objects is creative. By invoking the *glMaterialfv()* function, the objects now can have a more realistic effects in the light, such as reflecting and diffusing the environmental lights.

4 User Guide

Open .sln file to open the in Visual Studio 2019, compile and run the program. All the dependency libraries are included in the project folder so no additional installations are required. Once the programming is running, user can move around the scene with 'WASD' keys on the keyboard and use mouse to change the viewing directions. Note that user can freely 'fly' in the program. Moreover, user can press key 'R' to speedup the rotation speed of the Ferris wheel and press key 'X' to reduce the rotation speed of the Ferris wheel. User can also press key 'o' to switch between different sky effects.

[3]The *VectorClass* in my program is the open-source code from the book [5], which is available at: http://www.mathfor3dgameprogramming.com/code/VectorClasses.h

References

[1] "OpenGL Mathematics (GLM)," in *https://github.com/g-truc/glm*.

[2] "The OpenGL Extension Wrangler Library," in *http://glew.sourceforge.net/*.

[3] "The Free OpenGL Utility Toolkit," in *http://freeglut.sourceforge.net/*.

[4] J. T. Cannon and S. Dostrovsky, *The Evolution of Dynamics: Vibration Theory from 1687 to 1742*. Springer, New York, NY, first ed., 1981.

[5] E. Lengyel, *Mathematics for 3D Game Programming and Computer Graphics, Third Edition*. Boston, MA, USA: Course Technology Press, 3rd ed., 2011.

[6] J. Stam, "Real-time fluid dynamics for games," 05 2003.