Final thesis

Doodle Processing System Using Cinder Graphics and Bullet Physics

by

Zhang Wen

LITH-IDA-EX-2013/LIU-IDA/LITH-EX-A–12/074–SE

2013-01-22
Final thesis

Doodle Processing System Using Cinder Graphics and Bullet Physics

by

Zhang Wen

LITH-IDA-EX-2013/LIU-IDA/LITH-EX-A–12/074–SE

2013-01-22

Supervisor: Erik Berglund
Examiner: Erik Berglund
Abstract

This Master Thesis proposes a implementation for a doodle system which can scan hand drawn paper doodles using camera into the system and obtain a painting based on doodles impacting on a canvas in the system. The doodles which are scanned into the system are applied physical effects such as collisions and forces on. The implementation is supported by Cinder Library [1] and Bullet Physics Library [2].

The Cinder used in this thesis is an open source C++ library developed by The Barbarian Group. The doodle system uses multiple draw methods and the framework provided by Cinder to build graphic components and a new system in a quick and efficient way.

The Bullet Physics Library used in this thesis is a professional open source library for physics simulations. The Bullet is integrated into the doodle system as a component which perform physics simulations on doodles.
Acknowledgements

I would like to thank my examiner Erik Berglund for giving me this thesis opportunity. I appreciate the chance of participating this interesting project and the help he gave me through the whole thesis period. Finally I also want to thank my family and friends for listening to my endless talk about my master thesis.
# Contents

1 Introduction 1
   1.1 Objective .................................................. 1
   1.2 Feasibility .................................................. 1
   1.3 Limitations .................................................. 1
   1.4 Thesis Organization ....................................... 2

2 Background Study 3
   2.1 Cinder Library ............................................. 3
      2.1.1 Cinder App Program Flow ............................... 3
      2.1.2 Cinder Image Processing ............................... 4
   2.2 Bullet Physics Library .................................... 5
      2.2.1 Collision Detection ................................... 5
      2.2.2 Rigid Body Dynamics .................................. 6
   2.3 OpenGL Library ............................................. 6

3 Requirements List 8

4 Doodle System 10
   4.1 Configuration Phase ....................................... 12
      4.1.1 Drawing Algorithm Choose Phase ..................... 12
      4.1.2 Effect Algorithm Choose Phase ....................... 12
      4.1.3 Start Scan Phase ...................................... 13
      4.1.4 Menu System ........................................... 13
      4.1.5 Discussion About Singletons In System ............. 14
   4.2 Scan Phase .................................................. 15
      4.2.1 Start Camera Phase ................................... 16
      4.2.2 Scan Doodle Phase .................................... 16
      4.2.3 Start Draw Phase ...................................... 18
   4.3 Draw Phase .................................................. 18
      4.3.1 User Draw Phase ....................................... 18
      4.3.2 Start Effect Phase .................................... 18
   4.4 Painting Phase .............................................. 20
      4.4.1 Painting Phase ......................................... 20
      4.4.2 Start Scan and Reset Phase ........................... 25
4.5 Performance ......................................................... 25

5 Further Development ............................................. 30
  5.1 Further Development About Drawing Algorithm Modules .... 30
  5.2 Further Development About Scan Multiple Doodles ......... 31
  5.3 Further Development About Effect Algorithm Module ....... 31
    5.3.1 EI’Beem Fluid Library ..................................... 31
    5.3.2 MSAFluid Library .......................................... 32
    5.3.3 Bullet SPH Fluid ........................................... 32
  5.4 Further Development About Color Mode ....................... 32

6 Conclusions ......................................................... 34
Chapter 1

Introduction

1.1 Objective

The objective of this thesis is to build a doodle system to perform that the user can scan a hand drawn doodle on white paper in natural light into the system, draw the doodle in different ways in the system and apply different effects on the doodle. The doodle affected by the effect will eventually fall onto a canvas in the system to result in a painting. During this process mentioned above, the user can take pictures by pressing specific keys on keyboard. The specific requirements are presented in Chapter 3.

1.2 Feasibility

As more and more graphic related requirements are needed to be fulfilled nowadays, a plenty of commercial or open source graphic engines such as OGRE, Cinder become available. Easy-to-use and powerful APIs’ provided by those graphic engines guarantee the graphic part implementation of the doodle system.

Like graphic engines, more and more physics engines such as Bullet, Open Dynamics Engine come out because of growing needs of physics simulations. APIs’ provided by those physics engines support the physics part implementation of the doodle system. In addition, integration between two different engines is not a new subject. In some area, for example, game development area, graphic engine and physics engine are always integrated to be a part of game engine.

1.3 Limitations

Since this project is merely the first version, only basic modules are implemented. This thesis mainly focus on this first version and only the modules
that have been implemented are included in this thesis. But the architecture of the project, relationships among modules and the structure of modules leave a extendable space for further development.

1.4 Thesis Organization

The rest of the thesis is organized as follows. Background study is presented in Chapter 2. Requirement list is presented in Chapter 3. In chapter 4 the implementation of thesis project is presented. The future developments are discussed in Chapter 5. The conclusions are presented in Chapter 6.
Chapter 2

Background Study

This project is developed on top of Cinder framework and Bullet Physics library. Cinder and Bullet is not the only combination. We choose Cinder and Bullet is because that they are open source libraries and they are relatively well maintained among open source libraries. In addition, Cinder provides a very clear, easy to use framework and supports direct OpenGL code.

2.1 Cinder Library

Cinder is a free, open source C++ library which provides a lot of functions and classes for programmers to fast create coding in graphics, audio, video, networking, image processing and computational geometry. In addition, Cinder is a cross-platform library which means the same code works under different platforms such as Mac OS, Windows, iPhone and iPad. Cinder is developed at The Barbarian Group as an internal project at the beginning. But now it is maintained and further developed by a community of programmers and artists. Lots of efforts is put on Cinder to make it familiar and intuitive to C++ programmers by developing on top of idioms and techniques used commonly by the C++ community.

2.1.1 Cinder App Program Flow

Cinder provides a very clear and easy to understand program flow for programmers. The program flow is presented in Fig. 2.1. PrepareSettings and setup modules are used for parameters initialization. The loop among event handlers, update and draw modules represents that these three modules are executed during each frame. Event handlers module is responsible for processing events triggered by users. Mouse events including mouse down, mouse up, mouse move, mouse drag, mouse wheel and key events such as key up, key down are provided by Cinder library. Update module is designed for
programmers to add new positions, color values, radius values etc of 2D images or 3D models. Draw module is the place for programmers to use draw functions which are available in Cinder to draw different images 2D or 3D. The last shutdown module is used to quit program according to shutdown event triggered by users. In order to build a project, programmers need to fill codes in each module of the flow. This program flow is also adopted by our thesis project.

![Cinder Program Flow](image)

**Figure 2.1: Cinder Program Flow**

### 2.1.2 Cinder Image Processing

Cinder is well known for providing strong 2D and 3D graphics capabilities. A lot of drawing methods are available in Cinder for programmers to use. Those drawing methods support not only drawing 2D images such as circle, rectangle, line, vector, 2D texture etc, but also drawing 3D models to the screen. Since OpenGL is integrated into Cinder, the drawing methods mentioned above are basically implemented by wrapping OpenGL codes inside. Except that, full-featured classes for OpenGL textures, FOBs, GLSL,
2.2. BULLET PHYSICS LIBRARY

Chapter 2. Background Study

VBOs, lights, materials and display lists are available in Cinder as well. It is very convenient for programmers who are not familiar with OpenGL programming to have a quick start. Because drawing methods in Cinder fulfill almost all requirements needed for making a graphic related project. It is also very convenient for those programmers who are good at OpenGL programming to customize their own drawing methods by wrapping OpenGL codes.

2D images on the filesystem can be drawn to the screen through Cinder. Image process in Cinder is presented in Fig. 2.2. A surface which lives on CPU is created from the result of loadImage(). And then a texture which lives on GPU is created from the surface or directly from loading image on the filesystem. The image is drawn to the screen finally by using drawing methods with texture as a argument. Surface, texture mentioned above are classes provided Cinder for manipulations such as filtering, replacing sections of bitmaps, or tweaking values of pixels.

TriMesh and VBOMesh are two mesh types in Cinder for rendering 3D models. The creation of TriMesh and VBOMesh needs vertex information, color information, texture coordinate information. But unlike TriMesh, VBOMesh stores all vertex data on the graphic card and allows programmers to modify the data on the graphic card without downloading all of it. In order to create a mesh in a quick way, Objloader class is brought in to create a TriMesh or VBOMesh from loading a OBJ format file on the filesystem.

2.2 Bullet Physics Library

Bullet Physics is open source physics library which provides 3D collision, soft body and rigid body simulations and published under the zlib license.

2.2.1 Collision Detection

Collision detection in Bullet is performed by a collision dispatcher iterating over each pair of collision objects and applying different collision algorithms corresponding to different types of collision objects involved to compute contact points.
Collision object is an object which contains a world transform and a collision shape in Bullet. A large variety of collision shapes is provided in Bullet. Basic collision shapes include box shape which is defined by half length of its sides, sphere shape which is a sphere based on its radius, capsule shape which is a capsule around the Y axis, cylinder shape which is a cylinder around the Y axis, cone shape which is a cone around Y axis. Except the basic shapes mentioned above, a compound shape is available in Bullet for users to combine multiple convex shapes into a collision shape which suits their purpose.

Collision filtering is a very useful mechanism in Bullet to ensure that only certain objects collide with each other. Bitwise masks are supported by Bullet as a way to decide if collision objects should collide with other collision objects. This is performed by during collision detection only when the mask matches the group of other collision objects, collision objects will receive collisions.

2.2.2 Rigid Body Dynamics

Rigid body is derived from collision object with addition of forces, mass, inertia, velocity and constraints to move objects with non-zero mass and inertia. There are three different rigid body types in Bullet. Dynamic rigid bodies are rigid bodies which have positive mass and updated its world transform during each simulation frame. Static rigid bodies have zero mass and are not able to move but can have collisions with others. Like static rigid bodies, kinematic rigid bodies have zero mass. But kinematic rigid bodies are able to be animated by the user.

The world transform represents the transform of a rigid body containing no scaling, shear etc. The world transform is based on calculation of the local inertia, given a mass.

In order to get the world transform of collision objects, motionstates are provided by Bullet. It is very beneficial for the user that use the motionstates. Because computation consumption is saved by only involving bodies that have moved into computation. In addition, a shift between graphic object and center of mass transform can be tracked by the user.

2.3 OpenGL Library

OpenGL (Open Graphics Library) [3] provides about 150 distinct commands for rendering 2D/3D computer graphics. It is implemented as a hardware-independent interface on multiple hardware platforms. OpenGL does not have any commands for getting user inputs and describing 3D models. But OpenGL allows the user to create their own models from a small set of points, lines and polygons.

Cinder provides a lot of wrappers to common OpenGL elements and OpenGL functions. In addition, Cinder includes drawing functions which
are implemented by wrapping OpenGL codes in. This is very convenient for the user to have a quick start with graphic programming. An abstraction layer is built by Cinder between the user and OpenGL. For users who are good at OpenGL or want to build up their own models for rendering, it is allowed to directly program with OpenGL code.
Chapter 3

Requirements List

In order to implement the doodle system described in Chapter 1, a plenty of functions need to be listed and implemented individually. In this Chapter, we split the doodle system into a long requirements list to be fulfilled. As long as all requirements on the list are achieved, the doodle system will be created. The requirements list is listed as follows:

1. Hand drawn doodles on white paper in natural light can be scanned into the system by camera.

2. Doodles in the system are stored as a group of particles.

3. The user can draw scanned doodles on the screen by dragging mouse.

4. Multiple doodle drawing ways such as single and massive can be implemented.

5. Multiple effects such as physics effect can apply on doodles.

6. Multiple painting effects can be implemented such as oil painting, water color, 3D painting, sand simulation, fluid simulation etc.

7. The user can choose which doodle drawing way to apply.

8. The user can customize the chosen doodle drawing way by setting up corresponding parameters.

9. The user can choose which effect and painting effect to apply.

10. The user can customize the chose effect and painting effect by setting up corresponding parameters.

11. The user can observe the whole painting generation process from the screen.
12. The user can take picture during the painting generation process by pressing keyboard.

13. The user can choose to scan another doodle or set up parameters again without restarting the program after painting generated.

14. The doodle system can be installed on multiple platforms such as Windows, Mac, Ipad, Android etc.

In the end, requirements which are implemented are just a part of list above. Oil painting effect and 3D painting effect have performance problems, a further optimization will be in need. Fluid effect is not implemented, can be added to to-do list. Drawing massive doodles are not done. The doodle system can be installed only on Windows platform only.
Chapter 4

Doodle System

The doodle system consists of four subsystems which are module switching system, parameter module system, particle system and menu system. Module switching system is responsible for loading different modules according to different phases. Modules in the doodle system can be divided into configuration module, scan module, drawing algorithm modules, effect algorithm modules and other functional modules. Each module has at most one parameter module which stores all the parameters needed by the module. Similar to module switching system, parameter module system’s functionality is to load parameter modules for the user to set up the parameters in order to have a user desired performance. Particle system in the doodle system is in charge of creation, deletion, update, draw and a series of manipulations on particleObjs which is the data structure for scanned doodle and particles. Particles are the basic element of a particleObj. Like particle system, menu system supports creation, deletion, update, draw and a set of operations on menus and options in the system.

The program is always in one of four phases which are configuration phase, scan phase, draw phase and painting phase. The relations among phases is presented in Fig.4.1. When the doodle system starts running, it enters the configuration phase firstly, then switches to the scan phase, the draw phase and the painting phase. During the draw phase, the user can step back to the scan phase to scan a more satisfied doodle. During the painting phase, the user can choose to go to scan phase to scan another doodle or go back to the configuration phase to set up parameters and then restart.

The first phase is outlined in 4.1, the second phase is outlined in 4.2, the third phase is outlined in 4.3, the forth phase is outlined in 4.4 and the performance is outlined in 4.5.
Figure 4.1: Program Phase Flows
4.1 Configuration Phase

In this phase, a configuration module is loaded. This module is responsible for the user choosing drawing algorithms, effect algorithms and setting up corresponding parameters. Three child phases are split in configuration phase.

The first phase is outlined in 4.1.1, the second phase is outlined in 4.1.2, and the third phase is outlined in 4.1.3.

4.1.1 Drawing Algorithm Choose Phase

This phase is performed by the user choosing one drawing algorithm from a drawing algorithm selection menu on the screen. In what way the scanned doodle is drawn on the screen is determined by which drawing algorithm is chosen. Drawing algorithm here is a module type in the system. Only one drawing algorithm module is available in the current version. This module supports drawing a single doodle which is comprised of particles on the screen by the user dragging mouse in the draw phase. More drawing algorithm modules are able to be added into the system for the user choosing in the further development. More details can be found in Chapter 5.

After choosing the drawing algorithm module, a corresponding parameter module is activated for the user to set up parameters needed by the drawing algorithm module. Parameters are kept in the parameter module and used in the scan phase to create a user customized doodle. The parameter module mentioned above contains parameters such as properties of particle :particle radius, particle color, particle life span etc, properties of particle dip which is generated by a particle impacting onto the canvas in the system:dip resolution etc and color mode which represents color effect in the painting phase:default, oil painting and mesh.

This phase is implemented by that the configuration module sets the drawing flag via using an interface of drawing algorithm selection menu to draw the menu on the screen at the beginning. After the user choose the drawing algorithm, the parameter module contained in the option is activated and draws itself on the screen. Meanwhile, a drawing parameter confirm menu draws itself on the screen by the configuration module setting its drawing flag. When the parameter setting is done, the user click the confirm option in the drawing parameter confirm menu. The configuration module detects that the drawing flag of the drawing parameter confirm menu is false and sets the drawing flag of the effect algorithm selection menu to true to draw the menu on the screen. More details about option and menu can be found in Section 4.1.4.

4.1.2 Effect Algorithm Choose Phase

In this phase, an effect algorithm module is chosen from an effect algorithm selection menu by the user. What kind of effect which apply on the doodle
in the painting phase is decided by the chosen effect algorithm module. Only one physics effect algorithm module is offered in the current system. Collisions among particles and directional forces such as gravity are added on the doodle by this module. The effect algorithm selection menu is also extendable in the further development for the user to have more choices.

A corresponding parameter module is activated like in the drawing algorithm choose phase after the user choosing the effect algorithm module. A list of parameters such as gravity, anti-gravity, left force, right force and falling height is available for the user to set up. Parameters are kept in the module for the configuration in the painting phase.

In this phase, after the user choose the physics effect algorithm, the parameter module contained in the option is activated and draws itself on the screen. Meanwhile, a effect parameter confirm menu draws itself on the screen by the configuration module setting its drawing flag. When the parameter setting is done, the user click the confirm option in the effect parameter confirm menu. The configuration module detects that the drawing flag of the effect parameter confirm menu is false and sets the drawing flag of the start scan selection menu to true to draw the menu on the screen.

4.1.3 Start Scan Phase

This phase is performed by the user clicking a start scan option from the start scan selection menu. After clicking, the configuration module is ended and the scan module contained in the option is loaded in the module switching system. The doodle system enters the scan phase.

4.1.4 Menu System

The menu system in the doodle system consists of a menu controller module, multiple selection menus and multiple options. Menu controller is a module where all the selection menus in the system update and draw themselves. In addition, menu controller provides interface for creating selection menu on the run time. This is implemented by having a menu vector which stores all the menus in the doodle system in the menu controller. When new menus are created, they are pushed into the menu vector. During each frame, the menu vector iterates through every menus in the vector to update and draw those menus. The interface mentioned above which can create a selection menu on the run time supports further development of the doodle system. More detail is discussed in Chapter 5.

In order for the further development to customize menus, an abstract menu class is provided in the doodle system. Like menu controller, menu class contains a map which is used for all options of a menu to update and draw themselves. The first value in the map represents option which belongs to a menu, the second value in the map stores a menu which is a sub menu and can be drawn when mouse hovering on the option. The sub menu is not used in the current version, but it can be used in further development.
4.1. CONFIGURATION PHASE  CHAPTER 4. DOODLE SYSTEM

Different menus according to the specific requirements are able to be created by inheriting from the abstract menu.

There is only selection menu inherited from the abstract menu class in the current system. All the menus used in the system are selection menu objects. Except the functionality from abstract menu, selection menu provides interfaces corresponding to mouse events such as menu stopping drawing itself when a mouse clicking event on the selection menu occurs. Selection menu also provides interfaces for creating options and linking its related sub menu and interfaces for setting drawing flag and drawing position.

There are eight selection menus through whole system. Drawing algorithm selection menu contains a list of drawing options containing drawing algorithm modules which the user can choose to draw different doodles in the draw phase. Drawing parameter confirm menu only includes one option which is for the user to switch to effect algorithm selection menu. Like drawing algorithm selection menu, the user can select different effect algorithm modules to perform different effects on the doodle. Effect parameter confirm menu which also is comprised of one option switches to start scan menu by the user clicking. Start scan menu is used to load the scan module and end the configuration module. Start draw selection menu which is located in the scan phase also uses module switching system to enter the draw phase. Start effect selection menu and reset selection menu, like start draw selection menu above, are used to enter the painting phase and the configuration phase.

Options are basic elements in a menu. An option object contains at most one module reference and parameter module reference. There are three different types of options in the system. The first option type detects if the option is clicked and uses a change module interface from module switching system to load the module stored in the option. The second option type is used to pass the drawing algorithm module in the option to the start draw selection menu and use a change parameter module interface from parameter module system to load the parameter module in the option if there is one. Similar to the second type, the third option type passes the effect algorithm module contained in the option to the start effect selection menu and load its corresponding parameter module if it is null in the option. These three types mentioned above perform their functionality under different masks. In addition, option provides interfaces corresponding mouse events and interfaces for updating and drawing themselves. Option class also has interface for customizing according to different requirements such as different colors, different shapes, different width and height, different level of transparency etc.

4.1.5 Discussion About Singletons In System

Considering menu controller is needed across the system, menu controller is programmed according to Singleton pattern [4]. This is performed by
that only one object from instantiation of menu controller class through the whole system. Since menu controller keeps track of all menus in the doodle system, apparently it is not a good idea that menu controller is initialized to a new object every time it is needed.

The module switching system contains a module controller and multiple modules in the system. Module controller is responsible for loading modules and keeping track of the current running module. This is implemented by that there is a module vector in the module controller. When the module controller is loading a module, the old module on the top of the vector is popped out and new module is pushed into the vector. Module controller also provides an interface of loading modules for other components in the system. Since module controller is used in different options to load different modules in the system wide, it is more efficient if only one object of it exists.

Similar to the module switching system, the parameter module system consists of a parameter module controller and multiple parameter modules. The parameter module controller has the same functionality as the module controller except that modules are replaced by parameter modules which store all parameters needed by modules. For the same reason, the parameter module controller is also programmed to a singleton.

Configuration module, scan module, drawing algorithm modules, effect algorithm modules and other modules which will be introduced in later sections all inherit from a same abstract class named Algorithm. Considering modules will be loaded in the module controller multiple times, member variables of some modules are responsible for keeping track and member variables of some modules are needed in other components across the system, all modules are programmed as singletons. However, multiple instances from one module could be required in further development, maybe it is not a good idea to program all modules to singletons. Besides, too many global states in the system can result in difficulty of unit testing increasing and slow down performance because of multiple components fighting for the access to singleton. Only modules which act as service providers to other components are programmed to singleton can be an alternative.

Because of similar reasons, all parameter modules are made as singletons. Parameter modules act as data warehouse in the system. In the further development, synchronized blocks might be needed, singletons obviously will cause the system slowing down. Referring to the alternative mentioned above, it can be an idea to just make specific parameter modules as singletons.

### 4.2 Scan Phase

In scan phase, a hand drawn doodle on white paper is scanned into the system in natural light, and a doodle comprised of particles is created according to parameters of the chosen drawing algorithm module. There are three child phases in scan phase.
The first phase is outlined in 4.2.1, the second phase is outlined in 4.2.2 and the third phase is outlined in 4.2.3.

### 4.2.1 Start Camera Phase

In this phase, camera devices in the user’s computer are initialized and during each frame latest captures taken by camera devices are rendered on the screen. This is supported by a capture vector and a texture vector in the scan module.

Capture is a class in Cinder Library. It is used for obtaining all devices connected to the user computer system and manipulating those devices to capture video or return surfaces etc. Texture is a class in Cinder Library. It represents an image on the computer’s graphics card. As contrasted with texture, surface mentioned above is also a class in Cinder Library and represents an image located on the computer’s CPU.

When enters the scan phase, all devices connected to the user computer are listed out by using `ci::Capture::getDevices()`from Capture class and then are pushed into a device vector. The vector iterates through all elements and during each iteration one Capture object is created according to device and defined capture area and pushed into the capture vector. All Capture objects start capturing when they are pushed into the capture vector.

During each frame, latest surfaces returned by capture objects in the capture vector are transformed to corresponding texture objects and then pushed into the texture vector. Texture vector iterates through all textures in the texture vector and render those textures on the screen by using draw texture function from Cinder.

### 4.2.2 Scan Doodle Phase

This phase is performed by the user pressing a key on keyboard to scan a hand drawn doodle from camera. And a doodle which consists of multiple particles is created and stored in the particle system.

Except interfaces for updating and drawing, the scan module provides interface for scanning doodles. This interface responds to key events. When the user presses a certain key from the keyboard, a surface is captured and a channel object is created from this surface. Channel here is a class in Cinder. It represents an image in it’s own right - one for the red, the green, the blue and sometimes alpha. An grayscale channel is used in the system.

When the channel creation ends, a particleObj object is created by particle controller. Particle related parameters are assigned to the new created particleObj object for generating desired particles. Particle dip related parameters are also passed to the particleObj object for creating user customized painting. And then according color mode the user chosen, a different thresh value which controls maximum particles will be in the particleObj is initialized. Other parameters such as connectionDensity which determines distance among particles are used in scan algorithm.
Scan algorithm is a process of creating a doodle in the system from a hand drawn doodle. Scan starts by iterating through each pixel in the channel. The scan algorithm is presented in Fig 4.2. A central particle is created when the first pixel which fulfills the condition in the channel is met. The central particle is the central of a group of particles which form a particleObj object. All other particles which are created after central particle keep a relative distance to the central one. This mechanism is designed for the draw phase in which the user draws doodle by dragging mouse. A current mouse position is assigned to the central particle and all other particles inside particleObj object compute their own positions by adding relative distance with the position of central particle. As Fig.4.2 shows, two thresh values are imported into the scan algorithm. Thresh is a value for limiting the number of particles can be generated. This is designed for balancing between system performance and painting effect. The other thresh value is ConnectionDensity. It is used for arranging the density among particles. During the scan process, a maximum distance is obtained from computing every particle distance to the central particle. This maximum distance is needed by the particleObj object assigning slight varying colors to particles under default color mode.

Multiple particles, multiple particleObjs and a particle controller comprise the particle system. Particle controller keeps track of all particleObj objects by iterating through all elements in particleObj vector and provides interfaces for creating, updating and drawing particleObj objects. Besides, particle controller has interface for responding to mouse events such as moving a particleObj object by the user dragging mouse. Considering particle controller is used across whole system, it is designed as a singleton.

ParticleObj represents a doodle inside the doodle system. It manages particles which comprise the doodle and particle dips which form a painting. So there are two vectors in the particleObj. One is for keeping track of particles and the other one is for particle dips. In addition, particleObj provides interfaces for creating, updating, drawing particles and particle dips and interface for assigning varying colors similar to the color the user chose to particles in order to result in a better painting under default color mode and oil painting color mode. According to different color modes, corresponding blocks in particleObj are executed during the run time. More details about color mode can be found in Section 4.4.

Considering further extension, particleObj class has a name and id number which are designed for allowing the user scan and draw several doodles on the screen. Although it is not available in current version, it provides a support for further development. More details about further development is discussed in Chapter 5.

Particle class represents a basic element in the particle system. Particle provides interfaces for updating, drawing itself. Particle and particle dip mentioned above are instances of same class but under different type masks. Particle type masks is designed for managing different particle instances
in a easier way. There are four particle type masks in the current version. Sphere mask represents a sphere particle which is in XYZ coordinate system. Texture mask represents a circle particle on the XY plane. Mesh mask brings a 3D model particle with texture and shader. Dip mask determines a circle particle on the XZ plane. Particle type masks are determined by the color mode in the particleObj.

4.2.3 Start Draw Phase

This phase is performed by the user clicking a start draw option on start draw selection menu. After clicking the option, the scan phase is ended and the chosen draw algorithm module is loaded in the system.

There is a safe mechanism which is applied in the scan module. Only when a particleObj object is created from scanning, the start draw selection menu is drawn on the screen. This is considered because that in different light conditions, scans have possibilities to fail.

4.3 Draw Phase

In draw phase, the chosen drawing algorithm is applied and the user draws doodle by dragging mouse. Given the circumstance that only one drawing algorithm module is available in the current version, the user can only draw a single doodle in this phase. There are two child phases in draw phase.

The first phase is outlined in 4.3.1 and the second phase is outlined in 4.3.2.

4.3.1 User Draw Phase

In this phase, the chosen drawing algorithm has already loaded and the user draws a doodle by dragging mouse.

This is implemented by that when the user is dragging mouse, an interface for moving particleObj object in the particle controller is activated. This interface takes a current mouse position as argument and pass it to central particle in particleObj object. Then like we mentioned in 4.2 Section, all other particles in the particleObj object compute their own positions by adding central particle position with relative distance. After all these steps above, a doodle is drawn on the place where the user drags the mouse.

4.3.2 Start Effect Phase

This phase is performed by the user clicking a start effect option on start effect selection menu. After clicking the option, the draw phase is ended and the chosen effect algorithm module is loaded in the system.
4.3. DRAW PHASE

CHAPTER 4. DOODLE SYSTEM

```c
int con = 0;
while(!iters.line())
{
    if(-con >= mThresh)
        break;

    while(!iters.pixel())
    {
        if(-con >= mThresh)
            break;

        float gray = iters.v();
        if((gray/255) == 0.f)
            { if(-centralpos)
                {
                    p->addCentralParticle((iters.x()+translateX),( (iters.y()) + translateY));
                    this->mPixel.push_back(ci::Vec2f((iters.x()),( (iters.y()) )));
                    central = ci::Vec2f((iters.x()) + translateY, ( (iters.y()) + translateY));
                    centralpos = false;
                }
            } else
            {
                bool flag = false;
                for(vector<ci::Vec2f>::iterator p1 = this->mPixel.begin(); p1 != this->mPixel.end;
                {
                    float dis = p1->distance(iters.getPos());
                    if( dis <= (p->mRadius * 2 * ConnectionDensity) )
                        { flag = true; }
                }
            }
        if(!flag )
            { p->addParticles((iters.x()) + translateY),((iters.y()) + translateY));
              this->mPixel.push_back(ci::Vec2f((iters.x()),( (iters.y()) )));
              temps = ci::Vec2f( (iters.x()) + translateY, ( (iters.y()) + translateY ));
              float dis = temps.distance(central);
              if( dis >= maxdis )
                  maxdis = dis;
            }
        cont++;
    }
}
```

Figure 4.2: Scan Algorithm
4.4 Painting Phase

In painting phase, the chosen effect algorithm module which is physics effect algorithm module in this thesis is loaded and particles start to move based on forces applied on them. Then particles fall onto a canvas in the system and a painting is generated form it. During the painting phase, the user can take pictures via pressing a key on the keyboard. There are two child phases in effect phase.

The first phase is outlined in 4.4.1 and the second phase is outlined in 4.4.2.

4.4.1 Painting Phase

Physics effect algorithm module is implemented by integrating Bullet physics into Cinder. In the set up phase of this module, a dynamic world is created by using corresponding interfaces provided by the Bullet. Dynamic world is an environment where bullet physics simulation happen. The force parameters which are set up by the user and stored in the parameter module are passed into the dynamic world as the forces to apply on particles later.

Then a canvas object which is used for painting generation and a tracking camera object which is responsible for tracking the whole painting generation process are created. The particleObj object which represents a doodle in the system is then copied and pushed into physics stack which is a particleObj vector in the physics effect algorithm module. The mechanism here is designed to allow the user choose different doodles or multiple doodles at the same time to apply effects on. Although this mechanism is not used in the current version, it provides a support for the further extension. More details can be found in Chapter 5. After above steps are finished, rigid bodies are created based on particles’ and canvas’s radius, positions. Those rigid bodies are necessary components in the dynamic world in order to perform physics simulations. At the end of this set up phase, corresponding parameters which are used in two equations to compute splash radius when a particle hits on the canvas are initialized. The equations are presented below.

\[ W_e = \rho * u_0^2 * d_0 / \sigma \]  
\[ R = r_0 + 0.12 * \sqrt{W_e * r_0} \]

There are some certain assumptions [6] when using these two equations.

After the set up of the module is finished, simulation in the dynamic world is initiated. During each frame, positions of every particles and canvas are computed from simulation according to forces applied in dynamic world and which collision masks are turned on. Those positions are then assigned to corresponding particles and particles draw themselves based on new positions. Collision masks mentioned above is a mask mechanism in Bullet Physics designed to perform collision filtering. In order to have a basic doodle shape after particles falling on the canvas, three collision masks
are provided in the physics effect algorithm module. When particles are in flight, collision mask which represents collisions among particles is off, only collision mask which represents collisions between particle and canvas is on. Until each particle hits on the canvas, collisions with other particles is turned on. A splashing effect is resulted in from that.

When particles impact onto the canvas, particle dips are created at the place particles hit. This is implemented by using a particle dip creation interface from particleObj class. Dip radius is computed according to two equations presented before. Another mechanism here is color mode which is from the particleObj class. Different color modes decide not only what particle type will be used, but also what painting effect will be performed. There are three color modes available in the current version. Default color mode is the mode in which particle is drawn as a 3d sphere, particle color is adjusted similar to the color the user set up during configuration phase and particle dip is drawn as a 2d circle on XZ plane. The painting in default color mode is presented in Fig.4.3. Oil painting color mode is the second color mode in which particle is also drawn as a 3d sphere, particle dip is also drawn as a 2d circle on XZ plane. But a module based on oil painting algorithm [7] is executed during each frame to compute dip colors to result in oil painting effect. In order to achieve a clear oil painting effect, particle color is adjusted totally different to the color the user set up and each particle dip is created as a circle which is comprised of many small circles. The idea behind this is that considering a huge amount of computation is needed to calculate pixel colors in order to have oil painting effect, we decide to use a pixel simulation way to reduce the computation complexity. The specific way is that we make each dip formed by a group of smaller circles, instead of calculating pixels in each dip, we calculate colors of smaller circles. But this brings another problem here. Since draw circle method in Cinder library needs to pop matrix first and push back matrix after drawing, that means more circles are drawn more consumptions for popping and pushing are needed. While only one pop and push back matrix is needed if calculating pixels. In order to compare two solutions mentioned above, an analysis is certainly acquired here to get a trade off of how many circles comprise a dip. The painting in oil painting color mode is presented in Fig.4.4. The last color mode is 3D model color mode in which particle is drawn as a mesh, no color is needed and no particle dip is created when particle impacts onto canvas. The painting in 3D model color mode is presented in Fig.4.5. As Fig.4.5 shows, a 3D diamond model is used under 3D model color mode. It also supports other 3D models, for example when a sand model is used for particles, then a sand painting can be simulated on the screen.

A taking picture module is activated during the painting phase. This module is responsible for copying a window surface and saving as a picture responding to the user pressing key event.
Figure 4.3: Default color mode painting
Figure 4.4: Oil Painting Color Mode
Figure 4.5: 3D Model Color Mode
4.4.2 Start Scan and Reset Phase

This phase is performed by the user clicking a start scan option on start scan selection menu or a reset option on reset selection menu. Two selection menus are provided here. So the user can choose to scan another doodle without setting up parameters again. If the user is not satisfied by the parameters set up before, user can choose to go to configuration phase to start over.

4.5 Performance

In this section, we present the performance of the doodle system.

The hardwares the doodle system runs upon are: Intel(R) Core(TM) i3 CPU M330 @2.13GHZ, ATI Radeon HD 5470 graphic card and 2GB RAM memory. It is a old fashioned standard computer.

The performance is measured by FPS of the doodle system when different number of particles exist in the painting phase. More specifically, there are two kinds of particles in the painting phase. One particle is a 3D sphere particle which is a part of doodle in the system. And the other particle is a 2D circle particle which is created when a sphere particle impacts on the canvas in the system. Since only sphere particles have collision calculation consumption, circle particles have nothing but drawing consumption. The standard FPS in the doodle system is set to 30. That means frames will be updated 30 times per second. If the sphere particles’ collision calculation, drawing and circle particles’ drawing can not be done before next frame updating, next frame will be dropped. When FPS drops below 10, the painting phase will look fragmented.

As the number of sphere particle increases, FPS drops down gradually as Fig.4.6 shows. When sphere particle number is larger than 950, FPS drops below 20. Although more available sphere particles are better, more consumption on collision detection and drawing will lead to low FPS. And the fact that massive circle particles which are related to painting quality will result in FPS dropping also should be put into consideration. So we choose 950 as the maximum number of sphere particle allowed in the system.

When the consumption of sphere particles is certain, FPS of the doodle system drops down as the number of circle particle increasing. As Fig.4.7 shows, when the number of circle particle is larger than 24665, FPS drops below 10. If we set the maximum number of sphere particle to 950 and set the maximum number of circle particle to 24665, FPS remains above 10, the painting quality is also guaranteed as the result shows in Fig.4.8. and the whole painting phase is fluent from the user perspective. The performance is still acceptable.

It is the main reason leading to low performance that the Collision detection and drawing of particles consume too much time. Since collision detection is handled by Bullet, it is already optimized. In order to achieve
Figure 4.6: FPS By The Number Of Sphere Particles
Figure 4.7: FPS By The Number Of Circle Particles
Figure 4.8: Painting Under Certain Number of Particles
better performance, an optimization is needed for particle drawing. As mentioned in last section, when draws a particle, the matrix is popped and pushed once. One idea for optimization is to make the matrix popped and pushed only once for drawing all particles in the painting phase. In this way, consumption on particle drawing can be decreased, FPS will grow and more particles can be allowed to create in the painting phase.
Chapter 5

Further Development

In this chapter, we present some thoughts about further development of this project.

5.1 Further Development About Drawing Algorithm Modules

In the current version, there is only one drawing algorithm module available. A single doodle is drawn in the draw phase. More drawing algorithm modules can be added into the system. One idea is a drawing algorithm module which allows drawing massive doodles by the user dragging mouse. When the user drags mouse on the screen, a sphere comprised of scanned doodles or a random arranged group of doodles is drawn at the place mouse stays. This is can be implemented by using name and id member variables in particleObj class. During the scan phase, a particleObj object with a unique name is created and it’s id is initialized as 0. When enters the draw phase, multiple doodles with the same name but different id numbers are created in a certain organized group according to different algorithms. After the painting module is loaded, particleObj objects which have the same name are pushed into the gravity stack. A painting made by massive doodles can be generated.

Adding more draw algorithm modules to the system is also supported by the menu system. More drawing algorithm options can be added into drawing algorithm selection menu by using addoption interface from selection menu class. No matter what option in drawing algorithm selection menu is chosen by the user, the corresponding drawing algorithm module will be passed to start draw selection menu and parameters will be passed into the scan phase.
5.2 Further Development About Scan Multiple Doodles

Although multiple doodles can be scanned into the system, only the latest one is used in the draw phase. If multiple doodles are available, the user can have more choices to decide which doodle to draw or what combinations of different doodles to draw. And this can lead to very interesting painting result. One idea about how to implement this is presented here. A addmenu interface from menu controller class supports creating a selection menu during run time. This interface can be used in the scan phase. When a new particleObj object is created, a selection menu with one option which contains the chosen drawing algorithm module and the doodle name. Option class needs a bit modifications. Except loading new algorithm module, passing drawing algorithm module and passing effect algorithm module, a fourth functionality which can change doodle name in drawing algorithm module to doodle name in option object can be added. At the same time, a vector which stores all changed doodle names can be added in drawing algorithm module. When the user scan a new doodle, an option with doodle name is created and drawn on the screen. During the draw phase, the user can choose to draw different doodles by clicking corresponding options. The doodle name in drawing algorithm module is changed after each clicking, old one is saved to the vector. The doodle name is then passed to particle controller as a argument for particle controller to decide which doodle to draw. When effect algorithm module is loaded, the vector which have all drawn doodle names is passed to effect algorithm module in order to apply effect on those doodles.

5.3 Further Development About Effect Algorithm Module

During the development, we tried to add a fluid effect algorithm module which can make particles perform accurate fluid simulations. But it did not work well. Considering time limitation, we focus on integrating a open-source fluid library with the doodle system.

5.3.1 EI’Beem Fluid Library

The first candidate in our radar is an open-source free surface fluid library called EI’Beem[8]. It is based on the lattice-Boltzmann method (LBM) and is able to make fluid simulations very efficiently and physically accurate. EI’Beem is released under the GPL, and also integrated in 3D application Blender. But after visiting EI’Beem’s home page, we find out that EI’Beem stops further developing somehow and latest released version is available only for Unix like systems.
In order to integrate EI’Beem into the project which is on Windows platform, a lot of time is needed to make it available to Windows system or design a new fluid algorithm based on EI’Beem core. In the future development of the doodle system, it can be a possible way to implement fluid simulation that make EI’Beem available for Windows system.

5.3.2 MSAFluid Library

MSAFluid\[9\] comes in our sights after EI’Beem. MSAFluid is an open-source library and provides real-time fluid simulations which is based on Navier-Stokes equations. In addition, MSAFluid library is already integrated with Cinder library and is able to be used as a block under Cinder. It is very convenient to use MSAFluid with Cinder. However, MSAFluid provides operations on pixels only on XY-plane to implement real-time fluid simulations. And this does not match particle-based doodle system. Besides, fluid simulations performed by MSAFluid is more smoke trend than liquid trend.

5.3.3 Bullet SPH Fluid

There is a Smoothed Particle Hydrodynamics (SPH) fluid preview example in Bullet Physics library. It is particle based simulation which quite matches the particle based doodle system we are working on. But SPH fluid is not available in current Bullet version yet. It will be integrated into Bullet in near future. As we can imagine, it is a good choice to implement a fluid effect algorithm by using Bullet SPH fluid for the future development. Not to mention that integration complexity will be reduced by using SPH fluid, because SPH fluid will be a part of Bullet which has already integrated into the project.

5.4 Further Development About Color Mode

As mentioned in last Chapter 5, except default color mode other two color modes are in need of improvements. Huge amount of computations for particle oil painting colors during each frame is the first issue. Although a pixel simulation solution is applied, trade off between performance and number of small circles which simulate pixels is still unknown. For further development, an analysis is needed for comparison between computation consumption of direct pixel calculation and computation consumption of using pixel simulation and needed for obtaining a trade off between computer performance and number of small circles used in pixel simulation.

Low performance of 3D mesh rendering is the second issue. A limited small number of 3D particles are allowed to be created when 3D color mode is chosen. Computer performance goes down as thresh number of 3D particles increase. In order to achieve a high performance with huge amount of 3D particles, a optimization is apparently required in further development. The
problem can come from Cinder library itself. For example, Cinder does not support rendering 3D models in a massive way. The problem can also be caused by bad 3D particle creating and deleting policy.
Chapter 6

Conclusions

In this thesis, an extendable system which supports scanning a hand drawn doodle from the camera into the system and generating a painting by applying physics effect on the doodle has been implemented on top of Cinder and Bullet Physics libraries. This system fulfilled a part of requirements list. But there is still a part of requirements list remaining unimplemented. The system itself supports a further development by adding new modules into the existing system.

The performance of the doodle system on a standard computer is still acceptable. The scanned doodle size, the fluency of the painting phase and the quality of the painting are guaranteed. In order to achieve better performance, optimization for drawing particles is needed in the future development.
Bibliography


This Master Thesis proposes a implementation for a doodle system which can scan hand drawn paper doodles using camera into the system and obtain a painting based on doodles impacting on a canvas in the system. The doodles which are scanned into the system are applied physical effects such as collisions and forces on. The implementation is supported by Cinder Library [1] and Bullet Physics Library [2].

The Cinder used in this thesis is an open source C++ library developed by The Barbarian Group. The doodle system uses multiple draw methods and the framework provided by Cinder to build graphic components and a new system in a quick and efficient way.

The Bullet Physics Library used in this thesis is a professional open source library for physics simulations. The Bullet is integrated into the doodle system as a component which perform physics simulations on doodles.
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© [Zhang Wen]