Chapter 3 **Basic platform of the system**

In the first step the research develops the basic platform, which is extendable to contain every form of data and functions needed for integrating future media. Considering increasingly new media are being developed, this platform is developed as an open-source program to allow every developer to combine any new medium into the system. At this stage, the system only contains the basic functions and containers for combining media. The implementation of adding media into the system is done in latter steps. As shown in figure 3.1, the platform, main system database, GUI components, and output display are developed in this step. And the spaces and threads for processing media data storage, media data decode processes and media data analyze, synchronize, and transform processes are also arranged. Prototypes of system I/O, GUI, and database are developed in the first step. The platform is developed particularly to receive data from various media, combines them, and displays the result to inspire designers. Considering this system is an application that is extendable to contain different kinds of data and functionalities in the future. Also, the platform is designed with efficiency in mind, so that information from different media could be updated, synchronized, and displayed to reflect designers' changes in real time. Moreover, 3D data should be able to be manipulated and displayed since there is a deep relationship between 3D spaces and architectural design. Therefore, this research chooses C++ with OpenGL as the developing language. By using this open-source platform, other researchers can also add any media into this system by repeating the second step.

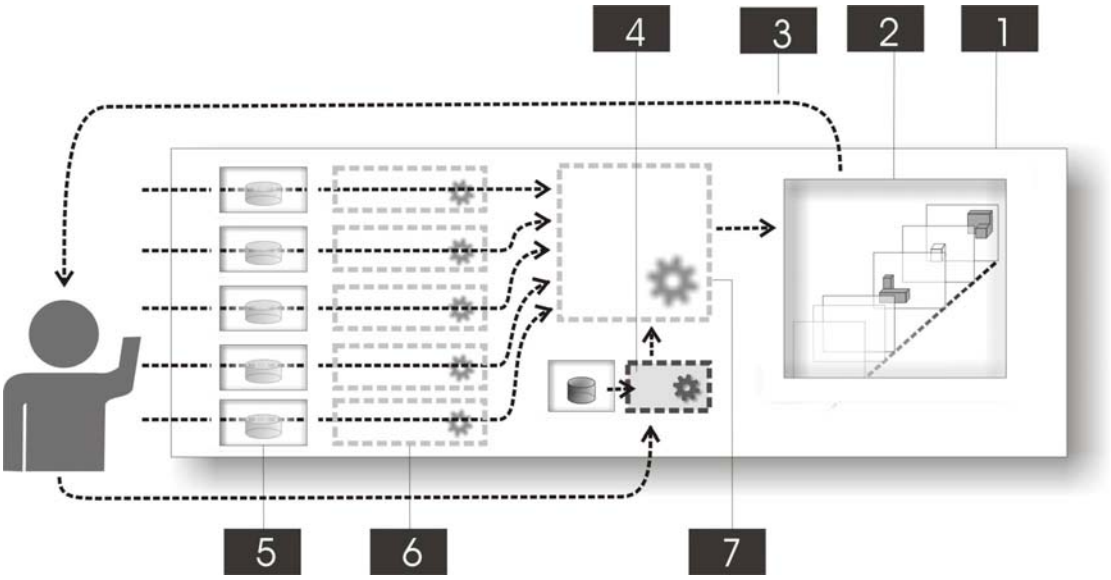

Figure 3.1: (1) the platform (2) main system database (3) output display (4) GUI component (5) media data storage (6) media data decode process (7) media data analyze, synchronize, and transform processes

In this chapter, the author first describes the form of the main output of this system, which is also related to the main database. Then, how the threads work in this system described. Last, the introduction of the GUI in this system is presented.

3.1 Output and database

Considering the most commonly used display devices display 2D images. Also, 2D images are compatible to sketches, which is one of the most commonly used design medium. Moreover, all media provides visual feedbacks for designers; visual feedback is one of the most important stimuli provided by media. Due to the above reasons, this research chooses 2D images as the form of the output display of this system. Note that this doesn't limit the variety of media data. Data besides 2D images, such as 3D models, can also be analyzed and synchronized with other media. As shown in figure 3.1.1, various types of data could be imported into the system and generate its visual feedback in a 2D form. Since the visual feedbacks of different media are compatible to each other, the system is able to combine them together.

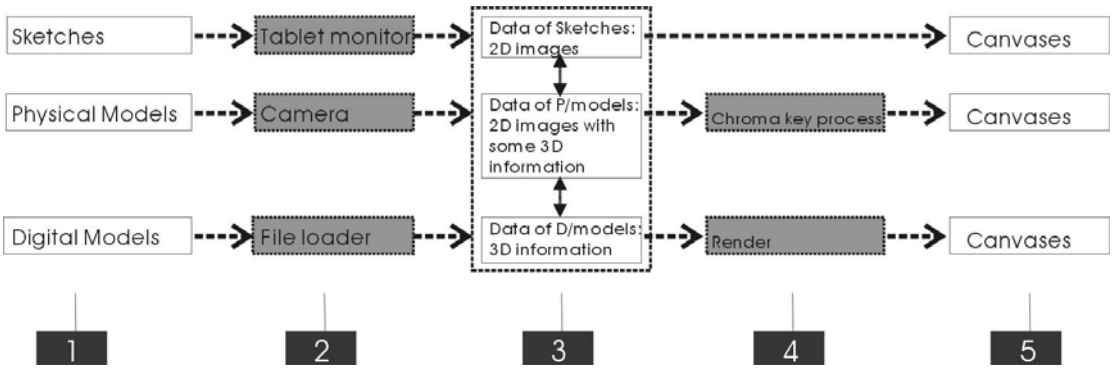

Figure 3.1.1: (1) media data (2) media data receive process (3) media data analyze and synchronize (4) media data transform (5) data storage (canvases)

Since the form of the system output is 2D image, the system's main database is designed to contain a series of canvases. The data of different media are processed, synchronized, transformed into the same form, and saved into individual canvas. Figure 3.1.1 shows the system running with three basic types of media: sketches, physical models, and digital models. All these three media could provide data more than 2D image for synchronizing. After the synchronizing, these media then be transformed into 2D images for combining and displaying. The implementations of integrating these three media are done in the second step of this research.

What designers will see is the combination of different canvases, which means the combination of various design media. A structure called "MyCanvas" is constructed first to contain canvas image data in the form of pixels. The formal idea is to store all the images into individual canvases and display them by their orders using OpenGL functions. However, images of some digital media, such as digital models, can be rendered directly using openGL. To be efficient, the system could directly draw these images in the frame buffer without storing them into canvas. Therefore, an abstractive structure "ASMCanvas" is constructed. Instead of storing the image, this structure stores an ID that indicates the rendering process of that canvas. Figure 3.1.2 shows how these two kinds of canvases work with two media, physical and digital models. Both media have a corresponding ASMCanvas but only physical models are connected to a MyCanvas. The system output displaying process renders each medium according to the order and the rendering process indicated by ASMCanvases.

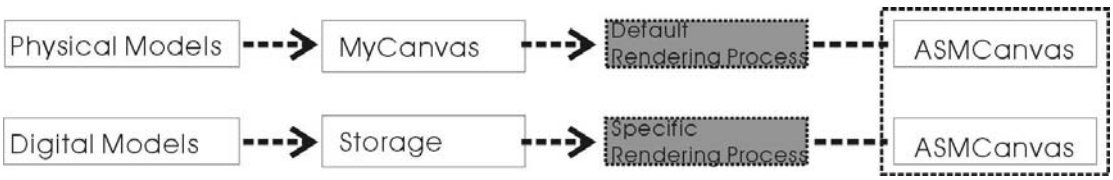

Figure 3.1.2: Physical models use both MyCanvas and ASMCanvas while Digital Models use only ASMCanvas.

The ASMCanvas structure contains only three variables: “type”, “ID”, and “visible”. The “type” is for indicating the rendering process while “ID” indicates another alternative parameter that can be used for indicating the function. Take the canvas that uses MyCanvas as example. The type of that ASMCanvas will points to the function that renders MyCanvas and the ID is used for passing to MyCanvas rendering function for indicating which MyCanvas should be rendered. The “visible” indicates if this canvas is visible or not. The ASMCanvas is also extensible to store data of each medium for the need of synchronizing. Other contents are added in latter steps and is described later. Structures of MyCanvas and ASMCanvas are listed below.

```
struct MyCanvas{
        COLORREF colorRGB[1024][768];
        int colorAlpha[1024][768];
};
struct ASMCanvas{
        int type;
        int ID;
        BOOL visible;
};
```

In conclusion, MyCanvas is constructed for the basic storage of media. The process for rendering MyCanvas is also constructed in this step. The system makes MyCanvas as a texture map and uses a quad to display it. MyCanvas provides only the basic needs and is independent from the system. The system use ASMCanvas to indicate the output rendering. Therefore developers are able to use any rendering method that is suitable for the medium. The code of the rendering function for MyCanvas is listed below.

```
void MyGL:: RenderMyCanvas(int id){
        glBindTexture(GL_TEXTURE_2D, texture_id[id]);
        setOrtho();
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glEnable(GL_BLEND);
        glDisable(GL_DEPTH_TEST);
        glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
        glBegin(GL_QUADS);
```

```
        glTexCoord2f(      0.0f, (float)15/32);
        glVertex3f(-2*windowHeight/3, -windowHeight/2,    0);
        ...
        glEnd();
        glDisable(GL_BLEND);
}
```

3.2 Threads

The basic processing framework is designed to be multi-threaded in order to receive different data from various design media simultaneously. The benefit of using a multi-threaded system is that the updating rate of each medium would not be affected by others. For example, if a web camera is used for capturing the images of physical models, a single-threaded system will be limited to the frame rate provided by the camera, which is usually about 30 frames-per-second. This kind of problem can be avoided by using a multi-threaded system. Moreover, the multi-threaded system separates the receiving of different media, which means it would be easier to add a new media or remove an existing one. Figure 3.2.1 shows three kinds of threads that maintain processing during the system running. All these threads process simultaneously. The threads for receiving media data are implemented when a medium is added into the system. Each medium has an individual thread that receives its data. At the same time, the synchronization thread keeps synchronizing the data from every media and sends the result to the rendering thread. While rendering thread renders the result, the data of each medium are updated by the synchronization thread for the next rendering. Therefore, the system would be able to display the results that reflect to any changes in real time.

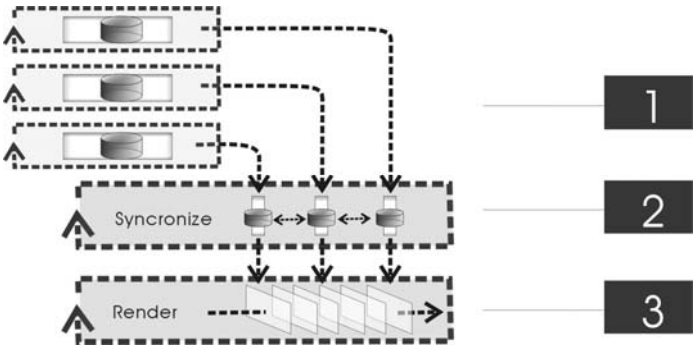

Figure 3.2.1: (1) Threads for receiving media data. (2) Thread for synchronize these data. (3) Threads for rendering the result.

This research uses GL utility tools (GLUT) to develop the system. GLUT is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. In the code using GLUT, glutDisplayFunc() is the function that is continuously processed. Its variable is a pointer to the function that actually being processed, which is renderScene(). Other functions that initialize the window and handle the inputs can also be seen in the following code. The code below shows how this research adopts GLUT to develop a multi-threaded system.

```
int main(int argc, char** argv){
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
        glutInitWindowPosition(0,0);
        glutInitWindowSize(640,480);
        AppGL.changeWinSize(1024,768);
        glutCreateWindow("ASM");

       //Main Looping Function
        glutDisplayFunc(renderScene);

       //Event Handling Functions
        glutReshapeFunc(changeSize);
        glutKeyboardFunc(processNormalKeys);
        glutSpecialFunc(processSpecialKeys);
        glutPassiveMotionFunc(processMousePassiveMotion);
        glutMotionFunc(processMouseMotion);
        glutMouseFunc(processMouse);

        AppGL.initGL();
        glEnable(GL_DEPTH_TEST);
        glutMainLoop();

        printf("end");
        system("PAUSE");
        return(0);
}
```

In renderScene(), which is in the code listed in the next session, we can see how this research adopts

GLUT to a multi-threaded platform that could suit the need of this research . A Boolean variable is used for each medium-receiving thread. The initialization of this variable is false. It will be set to true when the data receiving thread finished receiving. At the beginning of each loop, the system checks each of these variables. If the variable is true, which means the data is updated; the synchronizing thread will be activated.

```
void renderScene(void) {
        if(GLPoint->media1Updated){
                _beginthread(synchronize,0,NULL);
                GLPoint->media1Updated= false;
                _beginthread(UpdateMedia1,0,NULL);
        }
        ...
        AppGL.MyRenderScene();
}
```

In the basic actions of the receiving thread, the variable is set to false and the thread of receiving that medium is set to run again. Functions that actually receive the media data are implemented separately when a medium is added into the system (in the second step of this research). The code below shows the basic prototype of the receiving thread. In this code, updateMedia1() is the part that should be implemented separately to suit the needs of each medium.

```
void UpdateMedia1 ( void* pParams ) {
        updateMedia1();
        media1Updated = true;
        _endthread();
}
```

Since the synchronizing thread is implemented while adding the media into the system, it would be described in latter chapters. The displaying thread includes the displaying of ASMCanvases and GUI. In this thread, the system uses the orthographic mode with depth-buffer off. The system renders by the orders of ASMCanvases. The one at the bottom is first displayed. Each canvas and interface can be set to be visible or not. Considering most of the time designers may not want to use panels but to focus on the design, a global Boolean variable is used for indicating whether all interfaces are visible or not.

3.3 Graphic user interface

Prototype of basic tools such as GUI (graphic user interface) and file saving/loading are developed in this step. There are two main purposes for the GUI in this system. One is to let designers to control the parameters of the receiving media. Considering this system should be used by every designer in their own studio, the suitable parameters for receiving media could be different since the environment changed. Designers should be able to change these parameters to suit their working environment. The other purpose is to let designers to directly manipulate media. Although the main concept of this system is only to combine the results of different media, developers should also be able to directly add the medium on the system completely when it is a digital medium. In the second step, this research implements both these two kinds of media combining. Therefore, both two kinds of interfaces is implemented.

This research develops a set of GUI components which could be used to implement GUI for integrating different media. Menu bar, panels, and dialogues are developed. Most of the controls are made through the panels. The panel in this system contains most commonly used interface components that let developers to implement the GUI needed for the combining of each medium. As mentioned previously, designers (users) can use these interface to adjust the received media data or even directly manipulate the media. The structure for a simple panel is listed below.

```
struct panel{
    BOOL p_avalible;
    BOOL p_visible;
    BOOL p_selected;
    int p_width;
    int p_height;
    int p_x;
    int p_y;
    int p_mousex;
    int p_mousey;
    int buttonNum;
    int sliderNum;
    int stringNum;
    int plainImageNum;
    int colorImageNum;
    int textureImageNum;
        int layerTableNum;
    button *panelButton;
```

```
    slider *panelSlider;
    strings *panelString;
    plainImage *panelPI;
    colorImage *panelCI;
    textureImage *panelTI;
    layerTable *p_layerTable;
    int p_id;
    char *p_title;
    BOOL p_dragable;
    BOOL p_tagged;
    int p_order;
};
```

Main menu, dialogues, and one panel are implemented in this step. This panel is for designers to manipulate the canvases in the system database. Figure 3.3.1 shows the canvas panel and the main menu bar.

Figure 3.3.1: The Panel for designers to manipulate the canvases and the main menu for basic controls of the system

Unlike panels, dialogues provide short period information and input. Figure 3.3.2 shows the “open file dialogue” that is developed in this step. The result of this step (while no dialogue is activated) is shown in figure 3.3.3. The canvas panel and main menu bar can be seen in this figure. The system is only a container without content at this stage. Once the media are combined into the system, it will be able to display them.

Figure 3.3.2: The screenshot of the system while in dialogue mode.

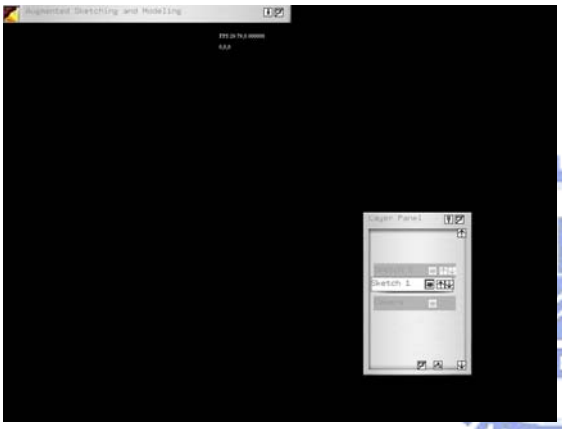

Figure 3.3.3: The display of the system at this stage.