วิชา: 523451 Computer Graphics

เรื่อง: PROJECT: OpenGL

สมาชิกในกลุ่ม

1. นายรชพล พงศ์กิตติศักดิ์ รหัสนักศึกษา B6216184

เนื้อหา

1. แหล่งที่มาของโค้ดที่นำมาปรับปรุงต่อยอด และแหล่งอ้างอิงอื่นๆ จำนวน 10 หัวข้อ
2. โค้ดส่วนที่พัฒนาขึ้นเอง พร้อมคำอธิบาย
   2.1 ไฟล์ phk_opengl.cpp คำอธิบาย 14 หัวข้อ
   2.2 ไฟล์ phk_opengl.h คำอธิบาย 2 หัวข้อ
3. รูปหน้าจอผลการทำงาน จำนวน 10 ภาพ

Project นี้สามารถ

1. หมุนมุมกล้องไปมาได้ โดยการคลิกแล้วลากเมาส์
2. เคลื่อนที่กล้องไปมาได้โดย

   W จะเคลื่อนที่ไปทางแกน +z

   S จะเคลื่อนที่ไปทางแกน -z

   A จะเคลื่อนที่ไปทางแกน +x

   D จะเคลื่อนที่ไปทางแกน -x

   Shift จะเคลื่อนที่ไปทางแกน -y

   Space bar จะเคลื่อนที่ไปทางแกน +y

   ซึ่งการเคลื่อนที่ของกล้องด้วนการกดปุ่มนี้จะไม่สัมพันธ์กับทิศทางที่เมาส์หันหน้าไป จะอ้างอิงการเคลื่อนที่ตามแกนเท่านั้น

3. สามารถกดปุ่ม R เพื่อที่จะกลับไปจุดเริ่มต้นได้เมื่อสบสนในการควบคุม โดยการกดปุ่ม R จะทำให้

   กล้องกลับไปที่ตำแหน่ง (1, 0, 1)

   กล้องเปลี่ยนการ zoom เป็นเหมือนครั้งแรกที่โปรแกรมเริ่มทำงาน

4. เลื่อน mouse wheel เพื่อให้กล้อง zoom in หรือ zoom out ได้

แหล่งที่มาของโค้ดที่นำมาปรับปรุงต่อยอด และแหล่งอ้างอิงอื่นๆ

1. ไฟล์ตัวอย่างของอาจารย์

2. Library ที่ใช้ในการโหลดไฟล์รูปภาพ

https://github.com/nothings/stb/blob/master/stb_image.h

3. วิธีการโหลด texture

https://www.youtube.com/watch?v=n4k7ANAFsIQ

4. วิธีการใส่ texture ให้วัตถุ

https://www.gamedeveloper.com/programming/understanding-and-using-opengl-texture-objects

5. วิธีการใส่แสง

https://www.glprogramming.com/red/chapter05.html

https://www.youtube.com/watch?v=oVwH8KV1xnY

6. วิธีการทำ mouse wheel input

https://docs.microsoft.com/en-us/windows/win32/inputdev/wm-mousehwheel

7. วิธีการทำ keyboard input

https://docs.microsoft.com/en-us/windows/win32/inputdev/about-keyboard-input

https://docs.microsoft.com/en-us/windows/win32/inputdev/using-keyboard-input

8. วิธีการหมุนวัตถุ

https://stackoverflow.com/questions/16578027/rotating-an-object-around-a-fixed-point-in-opengl

10. ต้นแบบของแผนที่

https://www.epicgames.com/fortnite/en-US/creative/island-codes/dust-2-counter-strike-8892-6024-6600

<u>โค้ดส่วนที่พัฒนาขึ้นเอง พร้อมคำอธิบาย</u>

ไฟล์ phk_opengl.cpp

1. ตัวแปรที่ใช้รองรับการนำเข้า texture

```
GLuint metal_door;
GLuint wood;
GLuint cement_road;
GLuint cement_wall;
GLuint sand;
GLuint brick;
GLuint building;
GLuint gate;
GLuint container;
GLuint container2;
GLuint box;
GLuint box2;
GLuint box3;
GLuint tank;
GLuint window;
GLuint french;
GLuint skyList[7];
```

2. ตัวแปรที่ใช้สำหรับการเคลื่อนที่ของกล้อง

```
GLfloat cameraPos[] = { 0.0, 0.5, 0.3};
GLfloat cameraFrontWS[] = { 0.0, 0.0, 1.0 };
GLfloat cameraFrontAD[] = { 1.0, 0.0, 0.0 };
GLfloat cameraFrontSHSP[] = { 0.0, 1.0, 0.0 };
GLfloat cameraUp[] = { 0.0, 1.0, 0.0 };
```

cameraPos คือ ตำแหน่งของกล้อง ณ ปัจจุบัน

cameraFrontWS คือ ทิศทางการเคลื่อนที่ของกล้องไปตามแกน z (เมื่อกดปุ่ม W หรือ S)

cameraFrontAD คือ ทิศทางการเคลื่อนที่ของกล้องไปตามแกน x (เมื่อกดปุ่ม A หรือ D)

cameraFrontSHSP คือ ทิศทางการเคลื่อนที่ของกล้องไปตามแกน y (เมื่อกดปุ่ม Shift หรือ Spacebar)

cameraUp คือ ทิศทางที่กล้องตั้งขึ้น (vector v)

3. function ที่ใช้สร้างพื้นเรียบ

```
void createPlane(
	float px, float py, float pz, float ws, float hs,
	int mode, float up[3], GLuint texture
) {
	int shape1[] = { // top and buttom mode 1
		0, 0, 0,
		0, 0, 1,
		1, 0, 1,
		1, 0, 0,
	};
	int shape2[] = { // side mode 2
		0, 0, 0,
		0, 0, 1,
		0, 1, 1,
		0, 1, 0,
	};
	int shape3[] = { // front & back mode 3
		0, 0, 0,
		1, 0, 0,
		1, 1, 0,
		0, 1, 0,
	};
	glEnable(GL_TEXTURE_2D);
	glBindTexture(GL_TEXTURE_2D, texture);
	glBegin(GL_POLYGON);
	glNormal3f(up[0], up[1], up[2]);
	for (int i = 0; i < 4 * 3; i += 3) { // ws = xs, hs = zs
		glTexCoord2f(shape1[i] + px, shape1[i + 2] + pz);
		switch (mode) {
			case 1:
				glVertex3f(
					ws * shape1[i] + px,
					shape1[i + 1] + py,
					hs * shape1[i + 2] + pz);
				break;
			case 2:
				glVertex3f(
					shape2[i] + px,
					hs * shape2[i + 1] + py,
					ws * shape2[i + 2] + pz);
				break;
			case 3:
				glVertex3f(
					ws * shape3[i] + px,
					hs * shape3[i + 1] + py,
					shape3[i + 2] + pz);
				break;
		}
	}
	glEnd();
	glDisable(GL_TEXTURE_2D);
}
```

3.1 input ของ function

```
void createPlane(
       float px, float py, float pz, float ws, float hs,
       int mode, float up[3], GLuint texture
)
```

px, py, pz คือ ตำแหน่งเริ่มต้นที่พื้นเรียบจะถูกสร้าง

ws คือ ความกว้างของพื้นเรียบ

hs คือ ความสูงของพื้นเรียบ

mode คือ รูปแบบของพื้นเรียบที่จะสร้าง

mode 1: พื้นเรียบที่ความกว้างอยู่ที่แกน x และความสูงอยู่ที่แกน z

mode 2: พื้นเรียบที่ความกว้างอยู่ที่แกน z และความสูงอยู่ที่แกน y

mode 3: พื้นเรียบที่ความกว้างอยู่ที่แกน x และความสูงอยู่ที่แกน y

up คือ normal vector ของพื้นเรียบ

texture คือ texture ที่ต้องการใส่ให้พื้นเรียบ

3.2 ตัวแปรที่ใช้กำหนดรูปแบบของพื้นเรียบ

```
int shape1[] = { // top and buttom mode 1
       0, 0, 0,
       0, 0, 1,
       1, 0, 1,
       1, 0, 0,
};
int shape2[] = { // side mode 2
       0, 0, 0,
       0, 0, 1,
       0, 1, 1,
       0, 1, 0,
};
int shape3[] = { // front & back mode 3
       0, 0, 0,
       1, 0, 0,
       1, 1, 0,
       0, 1, 0,
};
```

shape1 ใช้กับ mode 1

shape2 ใช้กับ mode 2

shape3 ใช้กับ mode 3

3.3 เรียกใช้ function ของ openGL ในการให้เริ่มใส่ texture

```
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texture);
```

glEnable(GL_TEXTURE_2D) เป็นการเริ่มใส่ texture

glBindTexture(GL_TEXTURE_2D, texture) เป็นการใช้ texture ที่ต้องการแปะลงบนพื้นเรียบ

3.4 เรียกใช้ function ของ openGL ในการให้เริ่มวาดพื้นเรียบ พร้อมกำหนด normal vector ของพื้นเรียบ

```
glBegin(GL_POLYGON);
glNormal3f(up[0], up[1], up[2]);
```

glBegin(GL_POLYGON) เป็นการเริ่มวาดพื้นเรียบโดยใช้รูปแบบของ polygon

glNormal3f เป็นการกำหนด normal vector ของพื้นเรียบที่วาด

3.5 algorithm ที่ใช้วาดพื้นเรียบ พร้อมกับ map texture ลงพื้นเรียบ

```
for (int i = 0; i < 4 * 3; i += 3) {
	glTexCoord2f(shape1[i] + px, shape1[i + 2] + pz);
	switch (mode) {
		case 1:
			glVertex3f(
				ws * shape1[i] + px,
				shape1[i + 1] + py,
				hs * shape1[i + 2] + pz);
			break;
		case 2:
			glVertex3f(
				shape2[i] + px,
				hs * shape2[i + 1] + py,
				ws * shape2[i + 2] + pz);
			break;
		case 3:
			glVertex3f(
				ws * shape3[i] + px,
				hs * shape3[i + 1] + py,
				shape3[i + 2] + pz);
			break;
	}
}
```

ใช้ for loop ทำซ้ำ 4 ครั้ง เท่ากับจำนวนจุดทั้งหมดของพื้นเรียบ

glTexCoord2f เป็นการกำหนดจุดที่จะใช้แปะ texture ลงบนพื้นเรียบ

glVertex3f เป็นการกำหนดจุดที่จะใช้วาดพื้นเรียบ

3.6 เมื่อวาดพื้นเรียบ และใส่ texture แล้ว

```
glEnd();
glDisable(GL_TEXTURE_2D);
```

glEnd เป็นการหยุดการวาดพื้นเรียบ

glDisable(GL_TEXTURE_2D) เป็นการหยุดการแปะ texture ลงบนพื้นเรียบ

4. function ที่ใช้สร้างพื้นเรียบแบบเอียง

```
void createPlaneSteep(
	float px, float py, float pz,
	float ws, float hs, int mode, float angle, float up[3], GLuint texture
) {
	int shape[] = { // top and buttom mode 1
			0, 0, 0,
			0, 0, 1,
			1, 0, 1,
			1, 0, 0,
	};
	glPushMatrix();
	glTranslatef(px, py, pz);
	switch (mode) {
		case 1:
			glRotatef(angle, 1.0f, 0.0f, 0.0f);
			break;
		case 2:
			glRotatef(angle, 0.0f, 1.0f, 0.0f);
			break;
		case 3:
			glRotatef(angle, 0.0f, 0.0f, 1.0f);
			break;
	}
	glTranslatef(-px, -py, -pz);
	glEnable(GL_TEXTURE_2D);
	glBindTexture(GL_TEXTURE_2D, texture);
	glBegin(GL_POLYGON);
	glNormal3f(up[0], up[1], up[2]);
	for (int i = 0; i < 4 * 3; i += 3) {
		glTexCoord2f(shape[i] + px, shape[i + 2] + pz);
		glVertex3f(ws * shape[i] + px, shape[i+1] + py, hs * shape[i+2] + pz);
	}

	glEnd();
	glDisable(GL_TEXTURE_2D);
	glPopMatrix();
}
```

4.1 input ของ function

```
void createPlaneSteep(
       float px, float py, float pz,
       float ws, float hs, int mode, float angle, float up[3], GLuint texture
)
```

px, py, pz คือ ตำแหน่งเริ่มต้นที่พื้นเรียบแบบเอียงจะถูกสร้าง

ws คือ ความกว้างของพื้นเรียบแบบเอียง

hs คือ ความสูงของพื้นเรียบแบบเอียง

mode คือ รูปแบบของพื้นเรียบแบบเอียงที่จะสร้าง

mode 1: พื้นเรียบจะถูกหมุนให้เอียงรอบแกน x

mode 2: พื้นเรียบจะถูกหมุนให้เอียงรอบแกน y

mode 3: พื้นเรียบจะถูกหมุนให้เอียงรอบแกน z

up คือ normal vector ของพื้นเรียบแบบเอียง

texture คือ texture ที่ต้องการใส่ให้พื้นเรียบแบบเอียง

4.2 ตัวแปรที่ใช้กำหนดรูปแบบของพื้นเรียบ

```
int shape[] = { // top and buttom mode 1
              0, 0, 0,
              0, 0, 1,
              1, 0, 1,
              1, 0, 0,
};
```

shape1 จะเป็นพื้นเรียบที่กว้างตามแกน x และสูงตามแกน z

function createPlaneSteep จะเป็นการวาดพื้นเรียบและตามด้วยการหมุนให้เอียง ในขณะที่ function createPlane จะเป็นการวาดพื้นเรียบตามรูปจุดที่กำหนดเอาไว้แล้ว

4.3 หมุนพื้นเรียบให้เอียงก่อนจะวาด

```
glPushMatrix();
glTranslatef(px, py, pz);
switch (mode) {
	case 1:
		glRotatef(angle, 1.0f, 0.0f, 0.0f);
		break;
	case 2:
		glRotatef(angle, 0.0f, 1.0f, 0.0f);
		break;
	case 3:
		glRotatef(angle, 0.0f, 0.0f, 1.0f);
		break;
}
glTranslatef(-px, -py, -pz);
```

glPushMatrix(px, py, pz) เป็นการเริ่มกระบวนการหมุน

glTranslatef เลื่อนจุดของพื้นเรียบออกไปเป็นระยะ px, py, pz

glRotatef ทำการหมุนจุดของพื้นเรียบตาม mode ที่ได้ใส่เข้ามา

glTranslatef(-px, -py, -pz) เลื่อนจุดของพื้นเรียบกลับที่เดิม

4.4 เรียกใช้ function ของ openGL ในการให้เริ่มใส่ texture

```
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texture);
```

glEnable(GL_TEXTURE_2D) เป็นการเริ่มใส่ texture

glBindTexture(GL_TEXTURE_2D, texture) เป็นการใช้ texture ที่ต้องการแปะลงบนพื้นเรียบ

4.5 เรียกใช้ function ของ openGL ในการให้เริ่มวาดพื้นเรียบ พร้อมกำหนด normal vector ของพื้นเรียบ

```
glBegin(GL_POLYGON);
glNormal3f(up[0], up[1], up[2]);
```

glBegin(GL_POLYGON) เป็นการเริ่มวาดพื้นเรียบโดยใช้รูปแบบของ polygon

glNormal3f เป็นการกำหนด normal vector ของพื้นเรียบที่วาด

4.6 algorithm ที่ใช้วาดพื้นเรียบ พร้อมกับ map texture ลงพื้นเรียบ

```
for (int i = 0; i < 4 * 3; i += 3) {
	glTexCoord2f(shape[i] + px, shape[i + 2] + pz);
	glVertex3f(ws * shape[i] + px, shape[i+1] + py, hs * shape[i+2] + pz);
}
```

ใช้ for loop ทำซ้ำ 4 ครั้ง เท่ากับจำนวนจุดทั้งหมดของพื้นเรียบ

glTexCoord2f เป็นการกำหนดจุดที่จะใช้แปะ texture ลงบนพื้นเรียบ

glVertex3f เป็นการกำหนดจุดที่จะใช้วาดพื้นเรียบ

เมื่อทำการวาดพื้นเรียบเรียบแล้วหมุนจะได้พื้นเรียบแบบเอียง

4.7 เมื่อวาดพื้นเรียบ และใส่ texture แล้ว

```
glEnd();
glDisable(GL_TEXTURE_2D);
glPopMatrix();
```

glEnd เป็นการหยุดการวาดพื้นเรียบ

glDisable(GL_TEXTURE_2D) เป็นการหยุดการแปะ texture ลงบนพื้นเรียบ

glPopMatrix เป็นการหยุดกระบวนการหมุน

5. function ที่ใช้สร้างทรงสี่หลี่ยม

```
void createCube(
	float px, float py, float pz, float xs, float ys, float zs, GLuint texture
) {
	int surface = 6;
	int shape[][16] = {
		{
			0, 0, 0,
			0, 1, 0,
			1, 1, 0,
			1, 0, 0,
		},
		{
			0, 0, 1,
			0, 1, 1,
			1, 1, 1,
			1, 0, 1,
		},
		{
			1, 0, 1,
			1, 0, 0,
			0, 0, 0,
			0, 0, 1,
		},
		{
			0, 1, 1,
			1, 1, 1,
			1, 1, 0,
			0, 1, 0,
		},
		{
			0, 0, 1,
			0, 1, 1,
			0, 1, 0,
			0, 0, 0,
		},
		{
			1, 1, 1,
			1, 0, 1,
			1, 0, 0,
			1, 1, 0,
		},
	};
	glEnable(GL_TEXTURE_2D);
	for (int i = 0; i < surface; i++) {
		glBindTexture(GL_TEXTURE_2D, texture);
		glBegin(GL_POLYGON);
		switch (i) {
			case 0: glNormal3f( 0.0,  0.0,  -1.0); break;
			case 1: glNormal3f( 0.0,  0.0,  1.0);  break;
			case 2: glNormal3f( 0.0,  1.0,  0.0);  break;
			case 3: glNormal3f( 0.0,  -1.0,  0.0); break;
			case 4: glNormal3f(-1.0,  0.0,  0.0);  break;
			case 5: glNormal3f( 1.0,  0.0,  0.0);  break;
			default: break;
		}
```

```
            for (int j = 0; j < 4 * 3; j += 3) {
                switch (i) {
                    case 0:
                    case 1: glTexCoord2f(
                                shape[i][j] + px,
                                shape[i][j + 1] + py);
                            break;
                    case 2:
                    case 3: glTexCoord2f(
                                shape[i][j] + px,
                                shape[i][j + 2] + pz);
                            break;
                    case 4:
                    case 5: glTexCoord2f(
                                shape[i][j + 2] + pz,
                                shape[i][j + 1] + py);
                            break;
                    }
                    glVertex3f(
                        xs * shape[i][j] + px,
                        ys * shape[i][j + 1] + py,
                        zs * shape[i][j + 2] + pz);
            }
            glEnd();
        }
        glDisable(GL_TEXTURE_2D);
}
```

5.1 input ของ function

```
void createCube(
        float px, float py, float pz, float xs, float ys, float zs, GLuint texture
)
```

px, py, pz คือ ตำแหน่งเริ่มต้นที่พื้นเรียบจะถูกสร้าง

xs คือ ความกว้างของพื้นเรียบ

ys คือ ความสูงของพื้นเรียบ

zs คือ ความยาวของพื้นเรียบ

texture คือ texture ที่ต้องการใส่ให้ทรงสี่หลี่ยม

5.2 ตัวแปรที่ใช้กำหนดจำนวนพื้นผิวของทรงสี่เหลี่ยม

```
int surface = 6;
int shape[][16] = {
	{
		0, 0, 0,
		0, 1, 0,
		1, 1, 0,
		1, 0, 0,
	},
	{
		0, 0, 1,
		0, 1, 1,
		1, 1, 1,
		1, 0, 1,
	},
	{
		1, 0, 1,
		1, 0, 0,
		0, 0, 0,
		0, 0, 1,
	},
	{
		0, 1, 1,
		1, 1, 1,
		1, 1, 0,
		0, 1, 0,
	},
	{
		0, 0, 1,
		0, 1, 1,
		0, 1, 0,
		0, 0, 0,
	},
	{
		1, 1, 1,
		1, 0, 1,
		1, 0, 0,
		1, 1, 0,
	},
};
```

surface คือ จำนวนพื้นผิวทั้งหมดของทรงสี่เหลี่ยม มีทั้งหมด 6 ด้าน

shape คือ ตัวแปรที่เก็บตำแหน่งการวาดจุดของพื้นผิวทั้ง 6 ด้าน

shape[0] คือ พื้นเรียบที่ความกว้างอยู่ที่แกน x และความสูงอยู่ที่แกน y (พื้นผิวด้านหน้า)

shape[1] คือ พื้นเรียบที่ความกว้างอยู่ที่แกน x และความสูงอยู่ที่แกน y (พื้นผิวด้านหลัง)

shape[2] คือ พื้นเรียบที่ความกว้างอยู่ที่แกน x และความสูงอยู่ที่แกน z (พื้นผิวด้านล่าง)

shape[3] คือ พื้นเรียบที่ความกว้างอยู่ที่แกน x และความสูงอยู่ที่แกน z (พื้นผิวด้านบน)

shape[4] คือ พื้นเรียบที่ความกว้างอยู่ที่แกน z และความสูงอยู่ที่แกน y (พื้นผิวด้านซ้าย)

shape[5] คือ พื้นเรียบที่ความกว้างอยู่ที่แกน z และความสูงอยู่ที่แกน y (พื้นผิวด้านขวา)

5.3 เรียกใช้ function ของ openGL ในการให้เริ่มใส่ texture และเริ่มวาดทรงสี่เหลี่ยม

```
glEnable(GL_TEXTURE_2D);
for (int i = 0; i < surface; i++) {
	glBindTexture(GL_TEXTURE_2D, texture);
	glBegin(GL_POLYGON);
	switch (i) {
		...
	}

	for (int j = 0; j < 4 * 3; j += 3) {
		...
	}
}
```

glEnable(GL_TEXTURE_2D) เป็นการเริ่มใส่ texture

ใช้ for loop ทำซ้ำ 6 ครั้ง เท่ากับจำนวนพื้นผิวทั้งหมดของทรงสี่เหลี่ยม

glBindTexture(GL_TEXTURE_2D, texture) เป็นการใช้ texture ที่ต้องการแปะลงบนพื้นผิวแต่ละด้าน

glBegin(GL_POLYGON) เป็นการเริ่มวาดพื้นเรียบโดยใช้รูปแบบของ polygon

5.4 กำหนด normal vector .ให้กับพื้นผิวแต่ละด้านของทรงสี่เหลี่ยม

```
switch (i) {
	case 0: glNormal3f( 0.0,  0.0, -1.0); break;
	case 1: glNormal3f( 0.0,  0.0, 1.0);  break;
	case 2: glNormal3f( 0.0,  1.0, 0.0);  break;
	case 3: glNormal3f( 0.0, -1.0,  0.0); break;
	case 4: glNormal3f(-1.0,  0.0, 0.0);  break;
	case 5: glNormal3f( 1.0,  0.0, 0.0);  break;
	default: break;
}
```

ในแต่ละ case ของคำสั่ง switch จะเป็นการกำหนด normal vector ให้กับพื้นผิวของทรงสี่เหลี่ยมตามตัวแปร i ที่เปลี่ยนไป โดย normal vector ทั้งหมดนี้จะชี้ไปด้านนอกของทรงสี่เหลี่ยม

5.5 algorithm ที่ใช้วาดพื้นผิวในแต่ละด้านของทรงสี่เหลี่ยม พร้อมกับ map texture ลงพื้นผิวในด้านนั้น

```
for (int j = 0; j < 4 * 3; j += 3) {
	switch (i) {
		case 0:
		case 1: glTexCoord2f(shape[i][j] + px, shape[i][j + 1] + py); break;
		case 2:
		case 3: glTexCoord2f(shape[i][j] + px, shape[i][j + 2] + pz); break;
		case 4:
		case 5: glTexCoord2f(shape[i][j + 2] + pz, shape[i][j + 1] + py); break;
	}
	glVertex3f(
		xs * shape[i][j] + px,
		ys * shape[i][j + 1] + py,
		zs * shape[i][j + 2] + pz
	);
}
```

ใช้ for loop ทำซ้ำ 4 ครั้ง เท่ากับจำนวนจุดทั้งหมดของพื้นผิวในแต่ละด้านของทรงสี่เหลี่ยม

ในแต่ละ case ของคำสั่ง switch จะใช้คำสั่ง glTexCoord2f กำหนดจุดที่จะใช้แปะ texture ลงบนพื้นผิว

glVertex3f เป็นการกำหนดจุดที่จะใช้วาดพื้นผิวแต่ละด้าน

5.6 เมื่อวาดพื้นผิว และใส่ texture แล้ว

```
for (int i = 0; i < surface; i++) {
	...
	...
	switch (i) {
		...
	}
	for (int j = 0; j < 4 * 3; j += 3) {
		switch (i) {
			...
		}
	}
	glEnd();
}
glDisable(GL_TEXTURE_2D);
```

glEnd เป็นการหยุดการวาดพื้นผิว

glDisable(GL_TEXTURE_2D) เป็นการหยุดการแปะ texture ลงบนพื้นผิว

6. function ที่ใช้สร้างโลกทรงสี่เหลี่ยม

```
void createWorld(
	float px, float py, float pz, float xs, float ys, float zs, GLuint texture[]
) {
	int surface = 6;
	int shape[][16] = {
		{
			0, 0, 0,
			0, 1, 0,
			1, 1, 0,
			1, 0, 0,
		},
		{
			0, 0, 1,
			0, 1, 1,
			1, 1, 1,
			1, 0, 1,
		},
		{
			1, 0, 1,
			1, 0, 0,
			0, 0, 0,
			0, 0, 1,
		},
		{
			0, 1, 1,
			1, 1, 1,
			1, 1, 0,
			0, 1, 0,
		},
		{
			0, 0, 1,
			0, 1, 1,
			0, 1, 0,
			0, 0, 0,
		},
		{
			1, 1, 1,
			1, 0, 1,
			1, 0, 0,
			1, 1, 0,
		},
	};
	glEnable(GL_TEXTURE_2D);
	for (int i = 0; i < surface; i++) {
		glBindTexture(GL_TEXTURE_2D, texture[i]);
		glBegin(GL_POLYGON);
		switch (i) {
			case 0: glNormal3f(0.0, 0.0, 1.0); break; //side
			case 1: glNormal3f(0.0, 0.0, -1.0);  break; //side
			case 2: glNormal3f(0.0, 1.0, 0.0);  break; // below
			case 3: glNormal3f(0.0, -1.0, 0.0); break; // above
			case 4: glNormal3f(1.0, 0.0, 0.0);  break; //side
			case 5: glNormal3f(-1.0, 0.0, 0.0);  break; //side
			default: break;
		}
```

```
		for (int j = 0; j < 4 * 3; j += 3) {
			switch (i) {
			case 0:
			case 1: glTexCoord2f(
					shape[i][j] + px,
					shape[i][j + 1] + py);
					break;
			case 2:
			case 3: glTexCoord2f(
						shape[i][j] + px,
						shape[i][j + 2] + pz);
					break;
			case 4:
			case 5: glTexCoord2f(
						shape[i][j + 2] + pz,
						shape[i][j + 1] + py);
					break;
			}
			glVertex3f(
					xs * shape[i][j] + px,
					ys * shape[i][j + 1] + py,
					zs * shape[i][j + 2] + pz
			);
		}
		glEnd();
	}
	glDisable(GL_TEXTURE_2D);
}
```

6.1 input ของ function

```
void createCube(
	float px, float py, float pz, float xs, float ys, float zs, GLuint texture
)
```

px, py, pz คือ ตำแหน่งเริ่มต้นที่พื้นเรียบจะถูกสร้าง

xs คือ ความกว้างของพื้นเรียบ

ys คือ ความสูงของพื้นเรียบ

zs คือ ความยาวของพื้นเรียบ

texture คือ texture ที่ต้องการใส่ให้โลกทรงสี่เหลี่ยม

6.2 ตัวแปรที่ใช้กำหนดจำนวนพื้นผิวของโลกทรงสี่เหลี่ยม

```
int surface = 6;
int shape[][16] = {
	{
		0, 0, 0,
		0, 1, 0,
		1, 1, 0,
		1, 0, 0,
	},
	{
		0, 0, 1,
		0, 1, 1,
		1, 1, 1,
		1, 0, 1,
	},
	{
		1, 0, 1,
		1, 0, 0,
		0, 0, 0,
		0, 0, 1,
	},
	{
		0, 1, 1,
		1, 1, 1,
		1, 1, 0,
		0, 1, 0,
	},
	{
		0, 0, 1,
		0, 1, 1,
		0, 1, 0,
		0, 0, 0,
	},
	{
		1, 1, 1,
		1, 0, 1,
		1, 0, 0,
		1, 1, 0,
	},
};
```

surface คือ จำนวนพื้นผิวทั้งหมดของโลกทรงสี่เหลี่ยม มีทั้งหมด 6 ด้าน

shape คือ ตัวแปรที่เก็บตำแหน่งการวาดจุดของพื้นผิวทั้ง 6 ด้าน

shape[0] คือ พื้นเรียบที่ความกว้างอยู่ที่แกน x และความสูงอยู่ที่แกน y (พื้นผิวด้านหน้า)

shape[1] คือ พื้นเรียบที่ความกว้างอยู่ที่แกน x และความสูงอยู่ที่แกน y (พื้นผิวด้านหลัง)

shape[2] คือ พื้นเรียบที่ความกว้างอยู่ที่แกน x และความสูงอยู่ที่แกน z (พื้นผิวด้านล่าง)

shape[3] คือ พื้นเรียบที่ความกว้างอยู่ที่แกน x และความสูงอยู่ที่แกน z (พื้นผิวด้านบน)

shape[4] คือ พื้นเรียบที่ความกว้างอยู่ที่แกน z และความสูงอยู่ที่แกน y (พื้นผิวด้านซ้าย)

shape[5] คือ พื้นเรียบที่ความกว้างอยู่ที่แกน z และความสูงอยู่ที่แกน y (พื้นผิวด้านขวา)

6.3 เรียกใช้ function ของ openGL ในการให้เริ่มใส่ texture และเริ่มวาดโลกทรงสี่เหลี่ยม

```
glEnable(GL_TEXTURE_2D);
for (int i = 0; i < surface; i++) {
	glBindTexture(GL_TEXTURE_2D, texture);
	glBegin(GL_POLYGON);
	switch (i) {
		...
	}

	for (int j = 0; j < 4 * 3; j += 3) {
		...
	}
}
```

glEnable(GL_TEXTURE_2D) เป็นการเริ่มใส่ texture

ใช้ for loop ทำซ้ำ 6 ครั้ง เท่ากับจำนวนพื้นผิวทั้งหมดของโลกทรงสี่เหลี่ยม

glBindTexture(GL_TEXTURE_2D, texture) เป็นการใช้ texture ที่ต้องการแปะลงบนพื้นผิวแต่ละด้าน

glBegin(GL_POLYGON) เป็นการเริ่มวาดพื้นเรียบโดยใช้รูปแบบของ polygon

6.4 กำหนด normal vector .ให้กับพื้นผิวแต่ละด้านของโลกทรงสี่เหลี่ยม

```
switch (i) {
	case 0: glNormal3f( 0.0,  0.0, -1.0); break;
	case 1: glNormal3f( 0.0,  0.0, 1.0);  break;
	case 2: glNormal3f( 0.0,  1.0, 0.0);  break;
	case 3: glNormal3f( 0.0, -1.0,  0.0); break;
	case 4: glNormal3f(-1.0,  0.0, 0.0);  break;
	case 5: glNormal3f( 1.0,  0.0, 0.0);  break;
	default: break;
}
```

ในแต่ละ case ของคำสั่ง switch จะเป็นการกำหนด normal vector ให้กับพื้นผิวของโลกทรงสี่เหลี่ยมตามตัวแปร i ที่เปลี่ยนไป โดย normal vector ทั้งหมดนี้จะชี้ไปด้านในของโลกทรงสี่เหลี่ยม

6.5 algorithm ที่ใช้วาดพื้นผิวในแต่ละด้านของโลกทรงสี่เหลี่ยมพร้อมกับ map texture ลงพื้นผิวในด้านนั้น

```
for (int j = 0; j < 4 * 3; j += 3) {
	switch (i) {
		case 0:
		case 1: glTexCoord2f(shape[i][j] + px, shape[i][j + 1] + py); break;
		case 2:
		case 3: glTexCoord2f(shape[i][j] + px, shape[i][j + 2] + pz); break;
		case 4:
		case 5: glTexCoord2f(shape[i][j + 2] + pz, shape[i][j + 1] + py); break;
	}
	glVertex3f(
		xs * shape[i][j] + px,
		ys * shape[i][j + 1] + py,
		zs * shape[i][j + 2] + pz
	);
}
```

ใช้ for loop ทำซ้ำ 4 ครั้ง เท่ากับจำนวนจุดทั้งหมดของพื้นผิวในแต่ละด้านของทรงสี่เหลี่ยม

ในแต่ละ case ของคำสั่ง switch จะใช้คำสั่ง glTexCoord2f กำหนดจุดที่จะใช้แปะ texture ลงบนพื้นผิว

glVertex3f เป็นการกำหนดจุดที่จะใช้วาดพื้นผิวแต่ละด้าน

6.6 เมื่อวาดพื้นผิว และใส่ texture แล้ว

```
for (int i = 0; i < surface; i++) {
	...
	...
	switch (i) {
		...
	}
	for (int j = 0; j < 4 * 3; j += 3) {
		switch (i) {
			...
		}
	}
	glEnd();
}
glDisable(GL_TEXTURE_2D);
```

glEnd เป็นการหยุดการวาดพื้นผิว

glDisable(GL_TEXTURE_2D) เป็นการหยุดการแปะ texture ลงบนพื้นผิว

7. function ที่ใช้สร้างรั้ว (สร้างทรงสี่เหลี่ยมหลายอัน)

```
void createfrench(
	float px, float py, float pz, float ws, float hs,
	int mode, int hfren,GLuint texture
) {
	for (float i = 0; i < ws; i += 2) {
		switch (mode) {
			case 1:
				createCube(px + i, py, pz, 1, hs, 1, texture);
				break;
			case 2:
				createCube(px, py, pz + i, 1, hs, 1, texture);
				break;
		}
	}
	switch (hfren) {
		case 1:
			switch (mode) {
				case 1:
					createCube(px, py + hs - 2, pz, ws, 1, 1, texture);
					createCube(px, py + 2, pz, ws, 1, 1, texture);
					break;
				case 2:
					createCube(px, py + hs - 2, pz, 1, 1, ws, texture);
					createCube(px, py + 2, pz, 1, 1, ws, texture);
					break;
			}
			break;
		default: break;
	}
}
```

7.1 input ของ function

```
void createfrench(
	float px, float py, float pz, float ws, float hs,
	int mode, int hfren,GLuint texture
)
```

px, py, pz คือ ตำแหน่งเริ่มต้นที่รั้วจะถูกสร้าง

ws คือ ความกว้างของรั้ว

hs คือ ความสูงของรั้ว

mode คือ รูปแบบของรั้วที่จะสร้าง

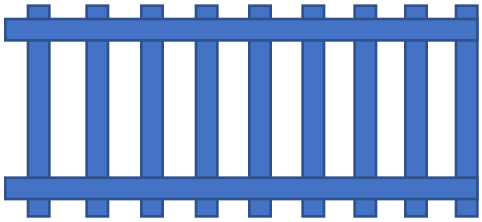
mode 1: รั้วที่ยาวออกไปทางแกน x

mode 2: รั้วที่ยาวออกไปทางแกน z

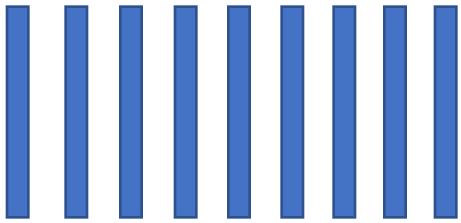
texture คือ texture ที่ต้องการใส่ให้รั้ว

hfren คือ ลักษณะของรั้วที่จะสร้าง

hfren 1: ให้เสารั้วมีการเชื่อมติดกันด้วยทรงสี่เหลี่ยมในแนวนอน



hfren เป็นเลขอื่น จะมีแค่เสารั้วเท่านั้น



7.2 algorithm ที่ใช้สร้างเสารั้ว

```
for (float i = 0; i < ws; i += 2) {
	switch (mode) {
		case 1:
			createCube(px + i, py, pz, 1, hs, 1, texture);
			break;
		case 2:
			createCube(px, py, pz + i, 1, hs, 1, texture);
			break;
	}
}
```

ใช้ for loop ทำซ้ำ เท่ากับจำความกว้างของรั้วที่กำหนด โดยเพิ่มค่าที่ละ 2 เพื่อให้เกิดช่องว่างระหว่างเสาของรั้ว createCube จะสร้างเสารั้วที่มีความกว้างตามแกน x 1 หน่วย และมีความกว้างตามแกน z 1 หน่วย โดยเสาจะสูงตามแกน y เท่ากับความยาว (hs) ที่ได้กำหนดให้กับ function นี้

7.3 algorithm ที่ใช้เชื่อมเสารั้วเข้าด้วยกัน

```
switch (hfren) {
	case 1:
		switch (mode) {
			case 1:
				createCube(px, py + hs - 2, pz, ws, 1, 1, texture);
				createCube(px, py + 2, pz, ws, 1, 1, texture);
				break;
			case 2:
				createCube(px, py + hs - 2, pz, 1, 1, ws, texture);
				createCube(px, py + 2, pz, 1, 1, ws, texture);
				break;
		}
		break;
	default: break;
}
```

switch(hfren) เมื่อ hfren เป็น 1 จะตรวจสอบค่าที่ตัวแปร mode

case 1: จะสร้างทรงสี่เหลี่ยมสูงตามแกน y 1 หน่วย และกว้างตามแกน z 1 หน่วย โดยจะกว้างตามแกน x เท่ากับความกว้าง (ws) ที่ได้กำหนดให้กับ function นี้

case 2: จะสร้างทรงสี่เหลี่ยมสูงตามแกน y 1 หน่วย และกว้างตามแกน x 1 หน่วย โดยจะกว้างตามแกน z เท่ากับความกว้าง (ws) ที่ได้กำหนดให้กับ function นี้

ถ้า switch(hfren) เมื่อ hfren เป็นเลขอื่นจะไม่มีการเชื่อมติดกันด้วยทรงสี่เหลี่ยมในแนวนอน

8. function ที่ใช้วาดวัตถุต่างๆเป็นรูปร่างรวมกัน

```
createWorld(-90, -200, -90, 320, 400, 320, skyList); // world

float planeUp[] = { 0.0, 1.0, 0.0 };
createPlane(-30, -30, -40, 75, 100, 1, planeUp, sand); // ground

createCube(1, -30, 1, 5, 30, 5, wood); // pillar1
createCube(40, -30, 1, 5, 30, 5, wood); // pillar2
createPlane(6, -30, 1, 34, 25, 3, planeUp, cement_wall); // wall1 attached to pillar1 and
2
createPlane(1, -5, 1, 44, 180, 1, planeUp, cement_road); // ceiling1 attached to pillar1
and 2
createPlane(1, -30, 6, 80, 25, 2, planeUp, cement_wall); // wall2 attached to pillar1
createCube(11, -30, 0, 10, 20, 1, metal_door); // door attached to wall at pillar1 and 2
createPlane(45, -30, 6, 20, 25, 2, planeUp, cement_wall); // wall3 attached to pillar2

createCube(40, -30, 26, 5, 30, 5, wood); // pillar3
createPlane(45, -45, 26, 45, 40, 3, planeUp, cement_wall); // wall4 attached to pillar3
createCube(85, -45, 6, 1, 30, 20, cement_wall); // chamber1 vertical part
createCube(85, -15, -40, 1, 10, 66, cement_wall); // chamber1 horizon part
createPlane(45, -5, 26, 41, 155, 1, planeUp, cement_road); // ceiling2 attached to wall4

createCube(15, -5, 100, 46, 20, 1, building);// wall5 upper ground
createCube(60, -5, 95, 1, 25, 5, building); // chamber2 horizon part1
createCube(60, -5, 65, 1, 20, 15, building); // chamber2 horizon part2
createCube(60, 15, 75, 1, 5, 20, building); // chamber2 vertical part
createCube(61, -5, 65, 50, 20, 1, building); // wall6 attached to chamber2 horizon part2
createCube(110, -5, 55, 1, 22, 10, building); // wall7 attached to wall6
createPlane(86, -5, -40, 70, 221, 1, planeUp, cement_road); // ceiling3 attached to wall6
and chamber1


createCube(107, -5, 35, 15, 35, 20, brick); // pillar_fortress1 attached to wall7
createCube(110, -5, -15, 5, 25, 50, brick); // wall_fortress1 attached to pillar_fortress1
createCube(115, 0, -15, 5, 25, 50, brick); // wall_fortress2 attached to pillar_fortress1
and wall_fortress1
createCube(107, -5, -35, 15, 35, 20, brick); // pillar_fortress2 attached to
wall_fortress1 and 2
createCube(122, -5, -30, 34, 30, 10, brick); // wall_fortress3 attached to
pillar_fortress2
createCube(122, -5, 40, 34, 30, 10, brick); // wall_fortress4 attached to pillar_fortress1
createCube(120, 16, -20, 36, 1, 60, brick); // wall_fortress5 attached to wall_fortress1
and 3
createCube(155, -5, -20, 1, 30, 60, brick); // wall_fortress6 attached to wall_fortress3
and 4


createCube(75, -5, 80, 50, 45, 30, building);// building_part1 near chamber2
createCube(125, -5, 80, 20, 60, 30, building);// building_part2 attached to building_part1
createCube(95, -5, 110, 30, 60, 30, building);// building_part3 attached to building_part1

createCube(100, -5, 140, 20, 50, 40, building);// building_part4 attached to
building_part3

float planeSteepUp[] = { 0.0, 1.0, 0.0 };
```

```
createPlaneSteep(-29, -30, 24, 30, 44, 1, -35,planeSteepUp, sand); // steep_ground1
attached to wall2
createPlane(-30, -5, 60, 31, 121, 1, planeUp, cement_road); // ceiling4 attached to
ceiling1 and steep_ground1
createPlaneSteep(45, -30, -40, 46, 66, 3, -19, planeSteepUp, sand); // steep_ground2
attached to ground
createPlane(88, -45, -40, 48, 131, 1, planeUp, sand); // lower ground


createCube(6, -5, 1, 34, 3, 5, cement_wall); // edge1 attached to pillar1 and 2
createCube(1, -5, 6, 5, 3, 54, cement_wall); // edge2 attached to pillar1
createCube(40, -5, 6, 5, 3, 20, cement_wall); // edge3 attached to pillar2 and 3
createCube(44, -5, 26, 40, 3, 5, cement_wall); // edge4 attached to pillar3
createCube(84, -5, -40, 5, 3, 71, cement_wall); // edge5 attached to edge4

createCube(-30, -46, -40, 186, 56, 1, building); // wall8 front of door
createCube(60, -45, -50, 20, 85, 20, building); // big_pillar1 attached to wall8

createCube(60, -5, 80, 1, 15, 15, gate); // gate attached to achamber2
creaeCube(-29, -5, 96, 44, 21, 20, container);// container1 on upper ground

createCube(10, -5, 20, 10, 10, 10, box); // box1 upper ground on ceiling1
createCube(22, -5, 16, 7, 7, 7, box); // box2 upper ground on ceiling1
createCube(80, -5, 55, 10, 10, 10, box); // box3 upper ground on ceiling1
createCube(97, -5, 55, 9, 9, 9, box); // box4 upper ground on ceiling1
createCube(81, 5, 56, 8, 8, 8, tank); // tank1 upper ground on box3
createCube(71, -5, 56, 8, 8, 8, tank); // tank2 upper ground near box3

// resize french1
glPushMatrix();
glTranslatef(61, 15, 65);
glScalef(0.5, 0.5, 0.5);
glTranslatef(-61, -15, -65);
createfrench(61, 15, 65, 100, 10, 1, 1, french); // french1 on wall6
glPopMatrix();

createfrench(109, 15, -20, 55, 1, 2, 0, wood); // french2 on wall_fortress1
createfrench(106, 25, 35, 20, 1, 2, 0, wood); // french3 on pillar_fortress1
createfrench(106, 25, -35, 20, 1, 2, 0, wood); // french4 on pillar_fortress2
createCube(110, 20, -20, 1, 1, 55,wood); // edge_wood1 attached to wall_fortress1
createCube(115, 25, -20, 1, 1, 55, wood); // edge_wood2 attached to wall_fortress2

createCube(107, 30, 35, 1, 2, 20, wood); // edge_wood3 (front) attached to
pillar_fortress1
createCube(121, 30, 35, 1, 2, 20, wood); // edge_wood4 (back) attached to pillar_fortress1
createCube(107, 30, 35, 15, 2, 1, wood); // edge_wood5 (left) attached to pillar_fortress1
createCube(107, 30, 54, 15, 2, 1, wood); // edge_wood46 (right) attached to
pillar_fortress1

createCube(107, 30, -35, 1, 2, 20, wood); // edge_wood7 (front) attached to
pillar_fortress1
createCube(121, 30, -35, 1, 2, 20, wood); // edge_wood8 (back) attached to
pillar_fortress1
createCube(107, 30, -35, 15, 2, 1, wood); // edge_wood9 (left) attached to
pillar_fortress1
createCube(107, 30, -16, 15, 2, 1, wood); // edge_wood10 (right) attached to
pillar_fortress1
```

```
createCube(135, -45, -40, 1, 40, 130, building); // wall_lower1 (back)
createCube(89, -45, 90, 47, 40, 1, building); // wall_lower1 (right)
createCube(89, -45, 26, 1, 40, 64, building); // wall_lower1 (left)

createCube(17, -5, 105, 40, 21, 20, container);// container2 (group) on upper ground
createCube(-10, 16, 105, 40, 21, 20, container2);// container3 (group) on container2

createCube(78, 18, 79, 10, 20, 1, window);// building_window1 on building_part1
createCube(95, 18, 79, 10, 20, 1, window);// building_window2 on building_part1
createCube(112, 18, 79, 10, 20, 1, window);// building_window3 on building_part1

createCube(124, 41, 98, 1, 12, 10, window);// building_window4 on building_part2
createCube(124, 41, 82, 1, 12, 10, window);// building_window5 on building_part2

createCube(97, 41, 109, 10, 12, 1, window);// building_window6 on building_part3
createCube(113, 41, 109, 10, 12, 1, window);// building_window7 on building_part3


createCube(99, 19, 143, 1, 20, 10, window);// building_window8 on building_part4
createCube(99, 19, 167, 1, 20, 10, window);// building_window10 on building_part4

createCube(74, 40, 79, 52, 1, 32, building); // edge11 (upper) on building_part1
createCube(74, 14, 79, 52, 1, 32, building); // edge12 (lower) on building_part1

createCube(124, 55, 79, 22, 1, 32, building); // edge13 (upper) on building_part2
createCube(124, 14, 79, 22, 1, 32, building); // edge14 (lower) on building_part2

createCube(94, 55, 109, 32, 1, 32, building); // edge15 (upper) on building_part3
createCube(94, 14, 109, 32, 1, 32, building); // edge16 (lower) on building_part3

        (99, 45, 139, 22, 1, 42, building); // edge17 (upper) on building_part3
createCube(99, 14, 139, 22, 1, 42, building); // edge17 (upper) on building_part3

createCube(-30, -46, -39, 1, 56, 139, building); // wall9 near container

glPushMatrix();
glTranslatef(72, -40, 15);
glRotatef(-20, 0.0, 0.0, 1.0);
glTranslatef(-72, 40, -15);
createCube(72, -40, 15, 10, 10, 10, box2); // box5 on steep_ground1
glPopMatrix();

glPushMatrix();
glTranslatef(78, -32, 16);
glRotatef(-20, 0.0, 0.0, 1.0);
glTranslatef(-78, 32, -16);
createCube(78, -32, 16, 5, 5, 8, box2); // box6 on box5
glPopMatrix();

createCube(0, -5, 125, 50, 21, 20, container2); // container4 (group) near building4
createCube(31, 16, 102, 20, 21, 50, container); // container5 (group) on container4 and 2
createCube(-23, -5, 119, 20, 21, 40, container2); // container6 (group) under  container4
createCube(-20, 16, 125, 40, 21, 20, container); // container7 (group) under  container5

createfrench(155, -5, 50, 130, 30, 2, 1, french); // french5 attached to wall_fortress3
createfrench(120, -5, 180, 36, 30, 1, 1, french); // french6 attached to french5

createfrench(-30, -5, 180, 130, 30, 1, 1, french); // french7 attached to building_part4
```

```
createfrench(-30, -5, 100, 82, 30, 2, 1, french); // french8 attached to french7

// rotate around y axis
glPushMatrix();
glTranslatef(123, -45, 76);
glRotatef(5, 0, 1, 0);
glTranslatef(-123, 45, -76);
createCube(123, -45, 76, 10, 10, 10, tank); // tank3 lower ground near wall
glPopMatrix();

// rotate around y axis
glPushMatrix();
glTranslatef(112, -45, 75);
glRotatef(-2, 0, 1, 0);
glTranslatef(-112, 45, -75);
createCube(112, -45, 75, 10, 10, 10, tank); // tank4 lower ground near tank3
glPopMatrix();


// rotate around y axis
glPushMatrix();
glTranslatef(91, -45, 75);
glRotatef(-2, 0, 1, 0);
glTranslatef(-91, 45, -75);
createCube(91, -45, 75, 9, 8, 9, box2); // box7 lower ground near tank4
glPopMatrix();


// rotate around y axis
glPushMatrix();
glTranslatef(91, -37, 77);
glRotatef(-2, 0, 1, 0);
glTranslatef(-91, 37, -77);
createCube(91, -37, 77, 5, 5, 5, box); // box8 lower on box7
glPopMatrix();

// rotate around y axis
glPushMatrix();
glTranslatef(102, -45, 76);
glRotatef(-2, 0, 1, 0);
glTranslatef(-102, 45, -76);
createCube(102, -45, 76, 10, 20, 10, box2); // box9 lower
glPopMatrix();

// rotate around y axis
glPushMatrix();
glTranslatef(101, -45, 70);
glRotatef(4, 0, 1, 0);
glTranslatef(-101, 45, -70);
createCube(101, -45, 70, 10, 5, 5, box3); // box9 lower
glPopMatrix();

// rotate around y axis
glPushMatrix();
glTranslatef(115, -45, 68);
glRotatef(-3, 0, 1, 0);
glTranslatef(-115, 45, -68);
createCube(115, -45, 68, 10, 5, 5, box3); // box9 lower
```

```
glPopMatrix();

createfrench(155, -5, -40, 10, 30, 2, 1, french); // french9 attached to
createCube(-30, -47, -40, 186, 1, 221, building); // cover1 (ground)
createCube(-30, -46, 180, 186, 41, 1, building); // cover2 (right)
createCube(155, -46, -40, 1, 41, 220, building); // cover3 (back)
createCube(-30, -46, 100, 1, 41, 80, building); // cover3 (front)
```

function draw นี้จะถูกเรียกใช้งานใน display เป็นการสร้างวัตถุด้วยการนำหลายชิ้นส่วนมาประกอบกันโดยใช้ function ที่สร้างขึ้นมา เพื่อทำให้การสร้างวัตถุต่างๆง่ายมากยิ่งขึ้น ส่วนรายละเอียดในการสร้างอ่านได้จาก comment สีเขียว

9. function ที่ใช้ update ค่าในการใช้ mouse wheel

```
void phkOpenGLengine::mouse_wheel_update(int nw)
{
	m_wheel = (nw + 1) % nw;
	switch (m_wheel) {
		case 1:
			m_nScoll++;
			break;
		default:
			m_nScoll--;
			break;
	}

	switch (m_nScoll) {
		case 3:
			m_nScoll = 4;
			break;
		case 11:
			m_nScoll = 10;
			break;
	}
}
```

9.1 input ของ function

```
void phkOpenGLengine::mouse_wheel_update(int nw)
```

nw คือ ค่าที่รับมาจากการใช้ mouse wheel

9.2 การคำนวณค่า m_wheel เพื่อแปลงค่า nw เป็นเลข 1

```
m_wheel = (nw + 1) % nw;
```

เมื่อเลขอะไรก็ตามมาบวก 1 แล้วหารแบบเอาเศษกับค่าเดิมของเลขนั้นจะได้ค่าเป็น 1 เสมอ

9.3 update ค่าที่แปลงมาเรียบร้อยแล้ว

```
switch (m_wheel) {
	case 1:
		m_nScoll++;
		break;
	default:
		m_nScoll--;
		break;
}
```

ตรวจสอบค่าที่ m_wheel

case 1: ให้ทำการบวกค่าของ m_nScoll (จำนวนครั้งในการหมุน mouse wheel) ไป 1 ค่า

ถ้าเป็นเลขอื่นๆ ให้ทำการลบค่าของ m_nScoll (จำนวนครั้งในการหมุน mouse wheel) ออก 1 ค่า

9.4 กำหนดขอบเขตที่สามารถหมุน mouse wheel ได้

```
switch (m_nScoll) {
	case 3:
		m_nScoll = 4;
		break;
	case 11:
		m_nScoll = 10;
		break;
}
```

ตรวจสอบค่าที่ m_nScoll

case 3: จะให้ค่า m_nScoll เท่ากับ 4

case 11: จะให้ค่า m_nScoll เท่ากับ 10

คำสั่งตรงนี้จะหมายถึง ไม่สามารถหมุน mouse wheel จนทำให้ตัวแปร m_nScoll มีค่าเกิน 11 และจะมีค่าไม่น้อยไปกว่า 3 (3 < m_nScoll < 11)

10. function ที่ใช้ในการ zoom กล้อง

```
void phkOpenGLengine::zoom() {
	float x, s;
	x = m_nScoll;
	s = x / 10.f;

	::glTranslatef(cameraPos[0], cameraPos[1], cameraPos[2]);
	::glScalef(s, s, s);
	::glTranslatef(-cameraPos[0], -cameraPos[1], -cameraPos[2]);

}
```

10.1 input ของ function

```
void phkOpenGLengine::zoom()
```

ไม่มี input ของ function แต่เป็นการเรียกใช้ function นี้ใน function display อีกที

10.2 การคำนวณอัตราการย่อ/ขยาย

```
Float x, s;
x = m_nScoll;
s = x / 10.f;
```

นำค่า m_nScoll มาหารด้วย 10 จะทำให้เป็นการย่อขนาดลง 1 / 10 ของขนาดเดิม

10.3 เรียกใช้ function ของ opneGL ในการย่อ/ขยายวัตถุ

```
::glTranslatef(cameraPos[0], cameraPos[1], cameraPos[2]);
::glScalef(s, s, s);
::glTranslatef(-cameraPos[0], -cameraPos[1], -cameraPos[2]);
```

เนื่องจากเป็นการย่อ/ขยายวัตถุรอบจุดอ้างอิงใดๆ จึงต้องทำการ

glTranslatef(cameraPos[0], cameraPos[1], cameraPos[2]) เลื่อนวัตถุไปด้วยระยะเท่ากับตำแหน่งของกล้อง

glScalef(s, s, s) ย่อ/ขยายตามขนาดที่คำนวณไว้แล้ว

glTranslatef(-cameraPos[0], -cameraPos[1], -cameraPos[2]) เลื่อนวัตถุกลับที่เดิม

11. function ที่ใช้ในการเคลื่อนที่ของกล้อง

```
void phkOpenGLengine::move(void) {
	gluLookAt(
		cameraPos[0], cameraPos[1], cameraPos[2],
		cameraPos[0]+cameraFrontWS[0],
		cameraPos[1] + cameraFrontWS[1],
		cameraPos[2] + cameraFrontWS[2],
		cameraUp[0], cameraUp[1], cameraUp[2]
	);
}
```

11.1 input ของ function

```
void phkOpenGLengine::move(void)
```

ไม่มี input ของ function แต่เป็นการเรียกใช้ function นี้ใน function display อีกที

11.2 เรียกใช้ function ของ opneGL ในการกำหนดตำแหน่งและการมองของกล้อง

```
gluLookAt(
	cameraPos[0], cameraPos[1], cameraPos[2],
	cameraPos[0]+cameraFrontWS[0],
	cameraPos[1] + cameraFrontWS[1],
	cameraPos[2] + cameraFrontWS[2],
	cameraUp[0], cameraUp[1], cameraUp[2]
);
```

camerPos คือ ตำแหน่ง ณ ปัจจุบันของกล้อง

cameraPos + cameraFrontWS คือ การคำนวณทิศทางที่กล้องต้องมองไป

cameraUp คือ vector v ที่ชี้ขึ้นด้านบนของกล้อง

12. function ที่ใช้ในการกำหนดค่าของแสง และโหลด texture เข้ามาใช้งาน

```
void phkOpenGLengine::initlightingAndTexture(void)
{
	//  initalize light
	GLfloat ambient[4] = { 1.0, 1.0, 1.0, 1.0 };
	GLfloat diffuse0[4] = { 1.0, 1.0, 1.0, 1.0 };
	GLfloat diffuse1[4] = { 1.5, 1.2, 1.2, 1.0 };
	GLfloat position0[] = { 4.0, 0.0, 8.0 , 1.0};
	GLfloat position1[] = { -1.0, 0.5, -5.0, 1.0 };

	GLfloat materialShininess[1] = { 8.0f };

	// enable all the lighting & depth effects
	::glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
	::glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
	::glLightfv(GL_LIGHT0, GL_POSITION, position0);

	::glLightfv(GL_LIGHT1, GL_AMBIENT, ambient);
	::glLightfv(GL_LIGHT1, GL_DIFFUSE, diffuse1);
	::glLightfv(GL_LIGHT1, GL_POSITION, position1);

	glShadeModel(GL_SMOOTH);
	::glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, materialShininess);

	::glEnable(GL_LIGHTING);
	::glEnable(GL_LIGHT0);
	::glEnable(GL_LIGHT1);

	::glEnable(GL_BLEND);
	::glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
	::glEnable(GL_LINE_SMOOTH);
	::glEnable(GL_NORMALIZE);


	int w;
	int h;
	int c;

	stbi_set_flip_vertically_on_load(true);
	unsigned char* image = stbi_load("./image/metalic_door.jpg", &w, &h, &c, 4);
	glGenTextures(1, &metal_door);
	glBindTexture(GL_TEXTURE_2D, metal_door);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
			GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture metalic_door");
		OutputDebugString(str);
	}
	stbi_image_free(image);
```

```
	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/dark_wood.jpg", &w, &h, &c, 4);
	glGenTextures(1, &wood);
	glBindTexture(GL_TEXTURE_2D, wood);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture dark_wood");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/cement_wall.jpg", &w, &h, &c, 4);
	glGenTextures(1, &cement_wall);
	glBindTexture(GL_TEXTURE_2D, cement_wall);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture cement_wall");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/cement_road4.jpg", &w, &h, &c, 4);
	glGenTextures(1, &cement_road);
	glBindTexture(GL_TEXTURE_2D, cement_road);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture cement_road4");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
```

```
	image = stbi_load("./image/brick.jpg", &w, &h, &c, 4);
	glGenTextures(1, &brick);
	glBindTexture(GL_TEXTURE_2D, brick);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture brick");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/sand.jpg", &w, &h, &c, 4);
	glGenTextures(1, &sand);
	glBindTexture(GL_TEXTURE_2D, sand);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture sand");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/cement_wall3.jpg", &w, &h, &c, 4);
	glGenTextures(1, &building);
	glBindTexture(GL_TEXTURE_2D, building);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture cement_wall3");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/gate.jpg", &w, &h, &c, 4);
```

```
	glGenTextures(1, &gate);
	glBindTexture(GL_TEXTURE_2D, gate);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture gate");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/container.jpg", &w, &h, &c, 4);
	glGenTextures(1, &container);
	glBindTexture(GL_TEXTURE_2D, container);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture container");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/box.jpg", &w, &h, &c, 4);
	glGenTextures(1, &box);
	glBindTexture(GL_TEXTURE_2D, box);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture box");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/water_tank.jpg", &w, &h, &c, 4);
	glGenTextures(1, &tank);
```

```
	glBindTexture(GL_TEXTURE_2D, tank);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture water_tank");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/window.png", &w, &h, &c, 4);
	glGenTextures(1, &window);
	glBindTexture(GL_TEXTURE_2D, window);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture window");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/iron_paint.jpg", &w, &h, &c, 4);
	glGenTextures(1, &french);
	glBindTexture(GL_TEXTURE_2D, french);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture iron_paint");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/dback.png", &w, &h, &c, 4);
	glGenTextures(7, skyList);
	glBindTexture(GL_TEXTURE_2D, skyList[0]);
```

```
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture dback");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/dfront.png", &w, &h, &c, 4);
	glBindTexture(GL_TEXTURE_2D, skyList[1]);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture dfront");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/dbuttom.png", &w, &h, &c, 4);
	glBindTexture(GL_TEXTURE_2D, skyList[2]);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture dbuttom");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/dtop.png", &w, &h, &c, 4);
	glBindTexture(GL_TEXTURE_2D, skyList[3]);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

```
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture dtop");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/dr.png", &w, &h, &c, 4);
	glBindTexture(GL_TEXTURE_2D, skyList[4]);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture dr");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/dl.png", &w, &h, &c, 4);
	glBindTexture(GL_TEXTURE_2D, skyList[5]);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture dl");
		OutputDebugString(str);
	}
	stbi_image_free(image);



	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/box2.jpg", &w, &h, &c, 4);
	glGenTextures(1, &box2);
	glBindTexture(GL_TEXTURE_2D, box2);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
```

```
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture box2");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/container2.jpg", &w, &h, &c, 4);
	glGenTextures(1, &container2);
	glBindTexture(GL_TEXTURE_2D, container2);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture container2");
		OutputDebugString(str);
	}
	stbi_image_free(image);


	stbi_set_flip_vertically_on_load(true);
	image = stbi_load("./image/box3.jpg", &w, &h, &c, 4);
	glGenTextures(1, &box3);
	glBindTexture(GL_TEXTURE_2D, box3);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
	glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
	if (image) {
		glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
				GL_UNSIGNED_BYTE, image);
	}
	else {
		sprintf(str, "cannot import texture box3");
		OutputDebugString(str);
	}
	stbi_image_free(image);
}
```

12.1 input ของ function

```
void phkOpenGLengine::initlightingAndTexture(void)
```

ไม่มี input ของ function แต่เป็นการเรียกใช้ function นี้ใน function initopengl อีกที

12.2 การกำหนดแสง

```
//  initalize light
GLfloat ambient[4] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat diffuse0[4] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat diffuse1[4] = { 1.5, 1.2, 1.2, 1.0 };
GLfloat position0[] = { 4.0, 0.0, 8.0 , 1.0};
GLfloat position1[] = { -1.0, 0.5, -5.0, 1.0 };

GLfloat materialShininess[1] = { 8.0f };

// enable all the lighting & depth effects
::glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
::glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
::glLightfv(GL_LIGHT0, GL_POSITION, position0);

::glLightfv(GL_LIGHT1, GL_AMBIENT, ambient);
::glLightfv(GL_LIGHT1, GL_DIFFUSE, diffuse1);
::glLightfv(GL_LIGHT1, GL_POSITION, position1);

glShadeModel(GL_SMOOTH);
::glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, materialShininess);

::glEnable(GL_LIGHTING);
::glEnable(GL_LIGHT0);
::glEnable(GL_LIGHT1);

::glEnable(GL_BLEND);
::glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
::glEnable(GL_LINE_SMOOTH);
::glEnable(GL_NORMALIZE);
```

ambient คือ ตัวแปรที่เก็บค่าสีของแสง ซึ่งเป็นแสงที่จะสว่างแม้ไม่มีแหล่งกำเนิดแสง

diffuse0 คือ ตัวแปรที่เก็บค่าสีของแสง ซึ่งเป็นแสงของแหล่งกำเนิดแสงที่ 0

diffuse1 คือ ตัวแปรที่เก็บค่าสีของแสง ซึ่งเป็นแสงของแหล่งกำเนิดแสงที่ 1

position0 คือ ตำแหน่งของแหล่งกำเนิดแสงที่ 0

position1 คือ ตำแหน่งของแหล่งกำเนิดแสงที่ 1

materialShininess คือ อัตราการสะท้อนแสงของวัตถุ

glLightfv เป็นคำสั่งใช้กำหนดคุณสมัติของแสงทั้ง ambient, diffuse และ position

glShadeModel(GL_SMOOTH) เป็นคำสั่งใช้ shade แบบ smooth

glMaterialfv เป็นคำสั่งใช้กำหนดคุณสมัติของวัตถุ โดยกำหนดเป็น shininess

glEnable(GL_LIGHTING) เป็นคำสั่งเปิดใช้งานแสง

glEnable(GL_LIGHT0) เป็นคำสั่งเปิดใช้งานแหล่งกำเนิดแสงที่ 0

glEnable(GL_LIGHT1) เป็นคำสั่งเปิดใช้งานแหล่งกำเนิดแสงที่

glEnable(GL_BLEND) เป็นคำสั่งเปิดใช้งาน blend

glBlendFunc เป็นการกำหนดรูปแบบของ blend ที่ใช้

glEnable(GL_LINE_SMOOTH) เป็นคำสั่งให้ใช้ line smooth

glEnable(GL_NORMALIZE) เป็นคำสั่งให้ใช้งาน normalize เป็นทิศทางที่แสงสะท้อนกับวัตถุ (normal vector)

12.3 การโหลด texture

```
int w;
int h;
int c;

stbi_set_flip_vertically_on_load(true);
unsigned char* image = stbi_load("./image/metalic_door.jpg", &w, &h, &c, 4);
glGenTextures(1, &metal_door);
glBindTexture(GL_TEXTURE_2D, metal_door);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
if (image) {
	glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, w, h, 0, GL_RGBA,
		GL_UNSIGNED_BYTE, image);
}
else {
	sprintf(str, "cannot import texture metalic_door");
	OutputDebugString(str);
}
stbi_image_free(image);
```

การโหลด texture จากไฟล์รูปภาพที่เก็บภายในเรื่องจะมีหลาย texture แต่ทุก texture จะใช้วิธีเดียวกันในการโหลดเหมือนกัน โดย

stbi_set_flip_vertically_on_load(true) เป็น flip vertical ภาพที่โหลดเข้ามาเพื่อไม่ให้ภาพกลับหัว

stbi_load("./image/file_name.jpg", &w, &h, &c, 4) เป็นการโหลดภาพในเครื่องตาม path ที่ให้

glGenTextures(1, &variable_name) เป็นการให้ตัวแปรที่กำหนดรับหน้าที่เก็บ texture

glBindTexture(GL_TEXTURE_2D, variable_name) เป็นการให้ตัวแปรที่กำหนดเก็บ texture แบบ 2D

glTexParameteri เป็นการกำหนดรูปแบบการ map texture ลงบนวัตถุ

จากนั้นให้ตรวจสอบว่าภาพนั้นถูกโหลดแล้วหรือไม่ ถ้า

โหลดได้สำเร็จ: ใช้คำสั่ง glTexImage2D ให้ภาพนั้นกลายเป็น texture

โหลดไม่ได้สำเร็จ: แสดงข้อความออกทาง console

จากนั้นเมื่อโหลดภาพเป็น texture เรียบร้อยแล้วให้คืนพื้นที่ความจำให้กับตัวแปรที่โหลดภาพเข้ามาจากเครื่อง

13. function WndProc (มี code แค่บ้างส่วนที่แก้ไขเพิ่มเติมเข้าไป)

```
LRESULT phkOpenGLengine::WndProc(UINT iMessage, WPARAM wParam, LPARAM lParam)
{
	float cameraSpeed = 2.0;

	switch (iMessage)
	{
	case WM_CREATE:
		break;

	case WM_TIMER:		if (m_isanimate)
	{
		m_rot[0] += 1.0f*(rand() / RAND_MAX) - 0.5f;
		m_rot[1] -= 1.0f*(rand() / RAND_MAX) - 0.5f;

		clamp(m_rot[0]);
		clamp(m_rot[1]);
		display();
	}
				break;

	case WM_PAINT:	display();
		break;

	case WM_SIZE:	resize(LOWORD(lParam), HIWORD(lParam));
		PostMessage(m_hWnd, WM_PAINT, 0, 0);
		break;

	case WM_LBUTTONDOWN:	SetCapture(m_hWnd);
		m_px = (float)LOWORD(lParam);
		m_py = (float)HIWORD(lParam);
		m_nDrag = 1;
		break;

	case WM_LBUTTONUP:	ReleaseCapture();
		m_px = 0.0f;
		m_py = 0.0f;
		m_nDrag = 0;
		break;

	case WM_MOUSEMOVE:
		if (m_nDrag) {
			int mx, my;
```

```
                mx = LOWORD(lParam);
                my = HIWORD(lParam);

                if (mx & (1 << 15)) mx -= (1 << 16);
                if (my & (1 << 15)) my -= (1 << 16);

                mouse_update((float)mx, (float)my);
                display();
        }
        break;
    case WM_MOUSEWHEEL:
        GET_WHEEL_DELTA_WPARAM(wParam);

        mouse_wheel_update(wParam);
        display();
        break;
    case WM_KEYDOWN:

        sprintf(str, "%c\n", wParam);
        OutputDebugString(str);
        switch (wParam)
        {

            case 'W':
                cameraPos[0] += cameraSpeed * cameraFrontWS[0];
                cameraPos[1] += cameraSpeed * cameraFrontWS[1];
                cameraPos[2] += cameraSpeed * cameraFrontWS[2];
                display();
                break;
            case 'S':
                cameraPos[0] -= cameraSpeed * cameraFrontWS[0];
                cameraPos[1] -= cameraSpeed * cameraFrontWS[1];
                cameraPos[2] -= cameraSpeed * cameraFrontWS[2];
                display();
                break;
            case 'A':
                cameraPos[0] += cameraSpeed * cameraFrontAD[0];
                cameraPos[1] += cameraSpeed * cameraFrontAD[1];
                cameraPos[2] += cameraSpeed * cameraFrontAD[2];
                display();
                break;
            case 'D':
                cameraPos[0] -= cameraSpeed * cameraFrontAD[0];
                cameraPos[1] -= cameraSpeed * cameraFrontAD[1];
                cameraPos[2] -= cameraSpeed * cameraFrontAD[2];
                display();
                break;
            case VK_SHIFT:
                cameraPos[0] -= cameraSpeed * cameraFrontSHSP[0];
                cameraPos[1] -= cameraSpeed * cameraFrontSHSP[1];
                cameraPos[2] -= cameraSpeed * cameraFrontSHSP[2];
                display();
                break;
            case VK_SPACE:
                cameraPos[0] += cameraSpeed * cameraFrontSHSP[0];
                cameraPos[1] += cameraSpeed * cameraFrontSHSP[1];
                cameraPos[2] += cameraSpeed * cameraFrontSHSP[2];
                display();
```

```
                        break;
                case 'R':
                        cameraPos[0] = 1;
                        cameraPos[1] = 0;
                        cameraPos[2] = 1;
                        m_nScoll = 10;
                        display();
                        break;
            }
            break;
            move();
        case WM_DESTROY:    purge();
            break;
        default:
            return DefWindowProc(m_hWnd, iMessage, wParam, lParam);
        }

        return 0;
}
```

13.1 input ของ function

```
LRESULT phkOpenGLengine::WndProc(UINT iMessage, WPARAM wParam, LPARAM lParam)
```

iMessage คือ message ที่รับเข้ามาใน application

wParam คือ ตัวแปรที่เกี่ยวข้องที่ส่งมาพร้อม message ซึ่งค่าที่ได้จะต่างกันออกไป และขึ้นอยู่กับการใช้งาน

lParam คือ ตัวแปรที่เกี่ยวข้องที่ส่งมาพร้อม message ซึ่งค่าที่ได้จะต่างกันออกไป และขึ้นอยู่กับการใช้งาน

13.2 mouse wheel input

```
case WM_MOUSEWHEEL:
        GET_WHEEL_DELTA_WPARAM(wParam);
        mouse_wheel_update(wParam);
        display();
        break;
```

เมื่อได้รับ message เป็น WM_MOUSEWHEEL จะใช้คำสั่ง GET_WHEEL_DELTA_WPARAM(wParam) นำค่า mouse wheel ไปเก็บไว้ใน wParam จากนั้นนำ wParam ไปใส่ function mouse_wheel_update เพื่อ update ค่าของ mouse wheel จากนั้นใช้คำสั่ง display เพื่อแสดงผลที่เปลี่ยนไปทันที

13.3 keyboard input

```
case WM_KEYDOWN:
	switch (wParam)
	{
		case 'W':
			cameraPos[0] += cameraSpeed * cameraFrontWS[0];
			cameraPos[1] += cameraSpeed * cameraFrontWS[1];
			cameraPos[2] += cameraSpeed * cameraFrontWS[2];
			display();
			break;
		case 'S':
			cameraPos[0] -= cameraSpeed * cameraFrontWS[0];
			cameraPos[1] -= cameraSpeed * cameraFrontWS[1];
			cameraPos[2] -= cameraSpeed * cameraFrontWS[2];
			display();
			break;
		case 'A':
			cameraPos[0] += cameraSpeed * cameraFrontAD[0];
			cameraPos[1] += cameraSpeed * cameraFrontAD[1];
			cameraPos[2] += cameraSpeed * cameraFrontAD[2];
			display();
			break;
		case 'D':
			cameraPos[0] -= cameraSpeed * cameraFrontAD[0];
			cameraPos[1] -= cameraSpeed * cameraFrontAD[1];
			cameraPos[2] -= cameraSpeed * cameraFrontAD[2];
			display();
			break;
		case VK_SHIFT:
			cameraPos[0] -= cameraSpeed * cameraFrontSHSP[0];
			cameraPos[1] -= cameraSpeed * cameraFrontSHSP[1];
			cameraPos[2] -= cameraSpeed * cameraFrontSHSP[2];
			display();
			break;
		case VK_SPACE:
			cameraPos[0] += cameraSpeed * cameraFrontSHSP[0];
			cameraPos[1] += cameraSpeed * cameraFrontSHSP[1];
			cameraPos[2] += cameraSpeed * cameraFrontSHSP[2];
			display();
			break;
		case 'R':
			cameraPos[0] = 1;
			cameraPos[1] = 0;
			cameraPos[2] = 1;
			m_nScoll = 10;
			display();
			break;
	}
	break;
```

cameraSpeed เป็นความเร็วของกล้องที่เคลื่อนที่ได้มีค่าเท่ากับ 2

เมื่อได้รับ message เป็น WM_KEYDOWN จะตรวจสอบค่าของ wParam ถ้า

case W: จะทำการบวกค่าของ cameraPos กับ cameraSpeed ที่คูณอยู่กับทิศทางที่กล้องจะเคลื่อนที่

ไป ในที่นี้เป็นการเคลื่อนที่ไปด้านหน้าในแกน z

case S: จะทำการลบค่าของ cameraPos กับ cameraSpeed ที่คูณอยู่กับทิศทางที่กล้องจะเคลื่อนที่ไป ในที่นี้เป็นการเคลื่อนที่ไปด้านหลังในแกน z

case A: จะทำการบวกค่าของ cameraPos กับ cameraSpeed ที่คูณอยู่กับทิศทางที่กล้องจะเคลื่อนที่ไป ในที่นี้เป็นการเคลื่อนที่ไปด้านซ้ายในแกน x

case D: จะทำการบวกค่าของ cameraPos กับ cameraSpeed ที่คูณอยู่กับทิศทางที่กล้องจะเคลื่อนที่ไป ในที่นี้เป็นการเคลื่อนที่ไปด้านขวาในแกน x

case shift: จะทำการบวกค่าของ cameraPos กับ cameraSpeed ที่คูณอยู่กับทิศทางที่กล้องจเคลื่อที่ไป ในที่นี้เป็นการเคลื่อนที่ไปด้านล่างในแกน y

case space: จะทำการบวกค่าของ cameraPos กับ cameraSpeed ที่คูณอยู่กับทิศทางที่กล้องจเคลื่อที่ไป ในที่นี้เป็นการเคลื่อนที่ไปด้านบนในแกน y

case R: จะทำการ set ค่าของ cameraPos .ให้เป็น (1, 0, 1) เพื่อให้กล้องไปอยู่ที่ตำแหน่งดังกล่าว และจากนั้นจะทำการ set ค่าของ m_nScoll เป็น 10 เพื่อให้อัตราการขยายกลับไปเท่าเดิมเหมือนครั้งแรกที่โปแกรมทำงาน

ทุก case ที่บรรทัดก่อนจะถึงคำสั่ง break จะใช้คำสั่ง display เพื่อให้แสดงผลทันทีที่กด keyboard

14. code ที่เพิ่มเติมเข้าไปส่วนที่ 1

```
LRESULT FAR PASCAL /*_export*/ phkOpenGLDefWndProc(HWND hwnd, UINT message, WPARAM
wParam, LPARAM lParam)
{
	phkOpenGLengine *pOpenGL = GetPointer(hwnd);
	SetFocus(hwnd);
	switch (message)
	{
	case WM_CREATE:
		if (!pOpenGL) {
			LPCREATESTRUCT lpcs;
			lpcs = (LPCREATESTRUCT)lParam;
			pOpenGL = (phkOpenGLengine *)lpcs->lpCreateParams;
			// Store a pointer to this object in the window's extra bytes;
			// this will enable us to access this object (and its member
			// functions) in WndProc where we are
			// given only a handle to identify the window.
			SetPointer(hwnd, pOpenGL);
			// Now let the object perform whatever
			// initialization it needs for WM_CREATE in its own
			// WndProc.
			SetFocus(hwnd);
			return pOpenGL->WndProc(message, wParam, lParam);
		}
		break;
	default:   if (pOpenGL) pOpenGL->WndProc(message, wParam, lParam);break;
	}
	return DefWindowProc(hwnd, message, wParam, lParam);
}
```

เพิ่ม SetFocus(hwnd) เข้าไปใน function phkOpenGLDefWndProc เพื่อให้หน้าต่างของโปรแกรมนี้สามารถรับ input ทาง keyboard ได้

14. code ที่เพิ่มเติมเข้าไปส่วนที่ 2

```
#define STB_IMAGE_IMPLEMENTATION
#include "./bsgl/stb_image.h"
#include <string.h> // for debug
```

มีการ include library เพิ่มเติม

stb_image ใช้สำหรับโหลดรูปภาพจากเครื่อง source code ของ library นี้ถูกใส่ไว้ใน project แล้ว

string ใช้สำหรับการ debug

มีการ define เพิ่มเติม

STB_IMAGE_IMPLEMENTATION ต้อง define ตามนี้ถ้าจะใช้งาน library stb_image

ไฟล์ phk_opengl.h

1. เพิ่ม method บางส่วนเข้าไปใน class phkOpenGLengine

```
protected:
        void    resize (int cx, int cy);
        void    mouse_update (float cx, float cy);
        void    initlightingAndTexture (void);
        void    recoverRigidDisplay (void);
        void    draw(void);
        void    zoom(void);
        void    mouse_wheel_update(int nw);
        void    move(void);
```

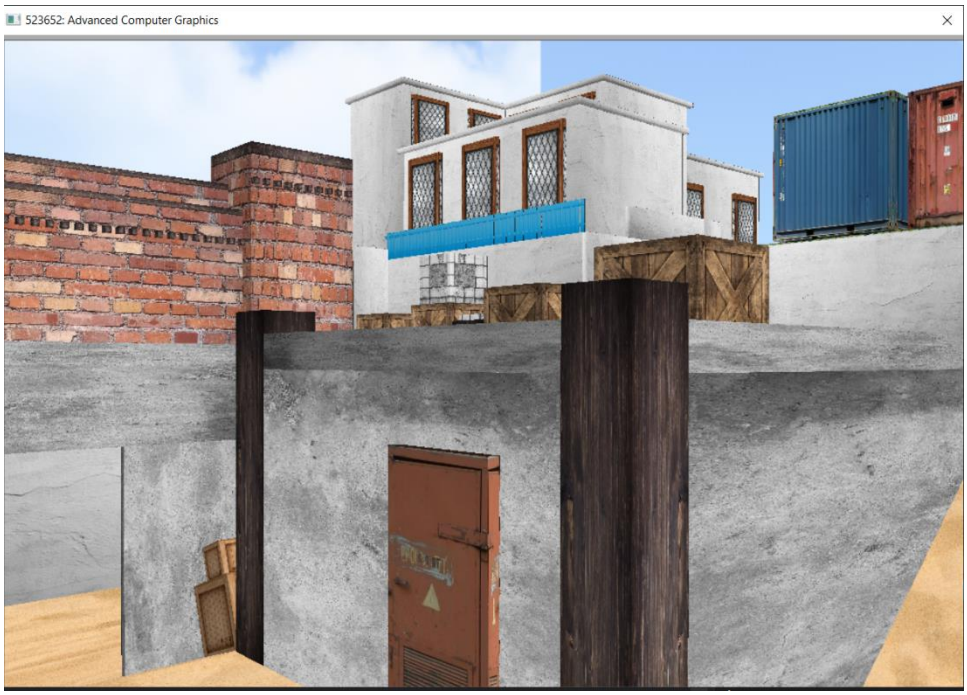
draw	method เดิมจะชื่อ drawaxes แค่เปลี่ยนชื่อเท่านั้น
zoom	method สำหรับการ zoom ของกล้อง
mouse_wheel_update	method สำหรับการ update ค่าของ mouse wheel โดยส่งค่า nw เข้าไป
move	method สำหรับการเคลื่อนที่ของกล้อง

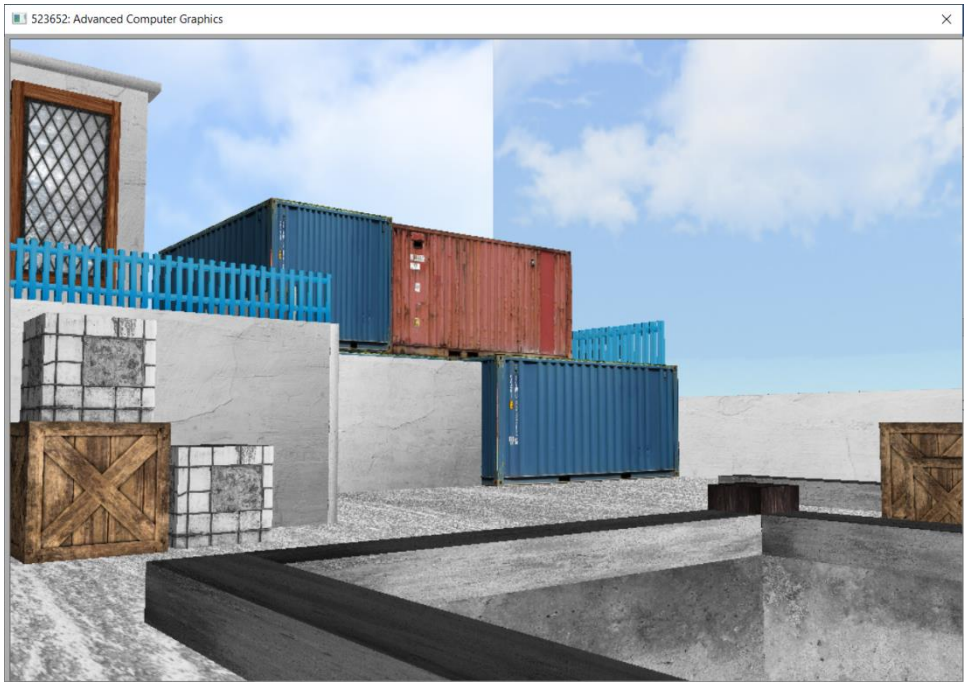
2. เพิ่มตัวแปรบางส่วนเข้าไปใน class phkOpenGLengine

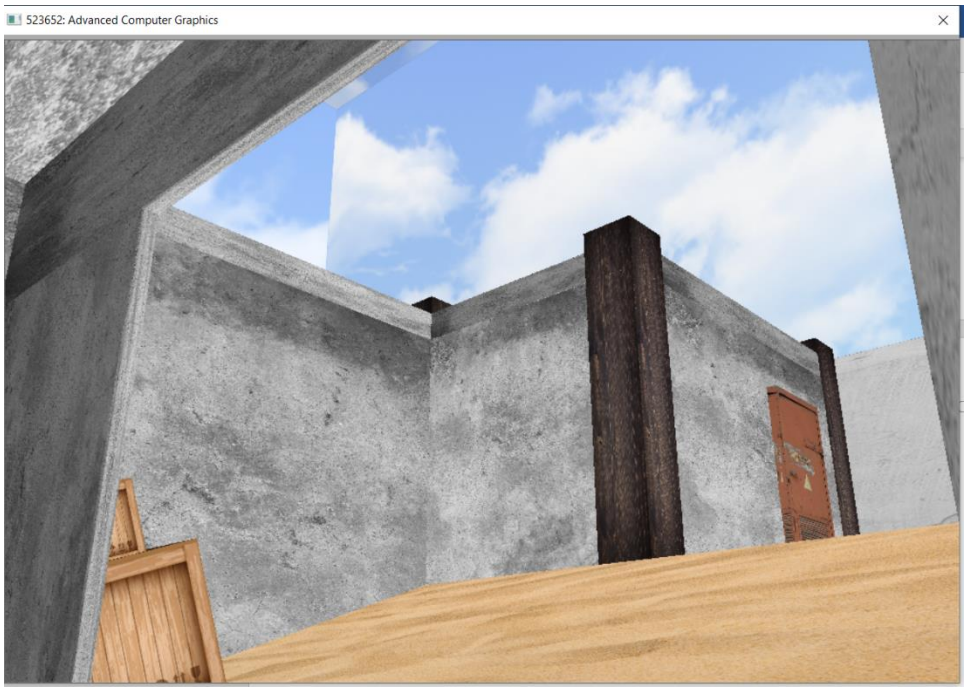
```
protected:
        HWND            m_hWnd, m_hParent, w_hWnd;
        int             m_nId, m_nDrag, m_wheel, m_nScoll = 10;
        ATOM            m_atom;
        HDC             m_hDC;
        HGLRC           m_hRC;
        float           m_rot [2], m_px, m_py;
        int             m_nListCreated [MAX_SURFACES];
        long            m_nmesh;
        float           m_fDir;
        int             m_isanimate;
```

m_wheel	คือ ตัวแปรที่เก็บค่าที่คำนวณมาจาก mouse wheel data
m_nScoll	คือ ตัวแปรที่เก็บจำนวนการหมุนของ mouse wheel

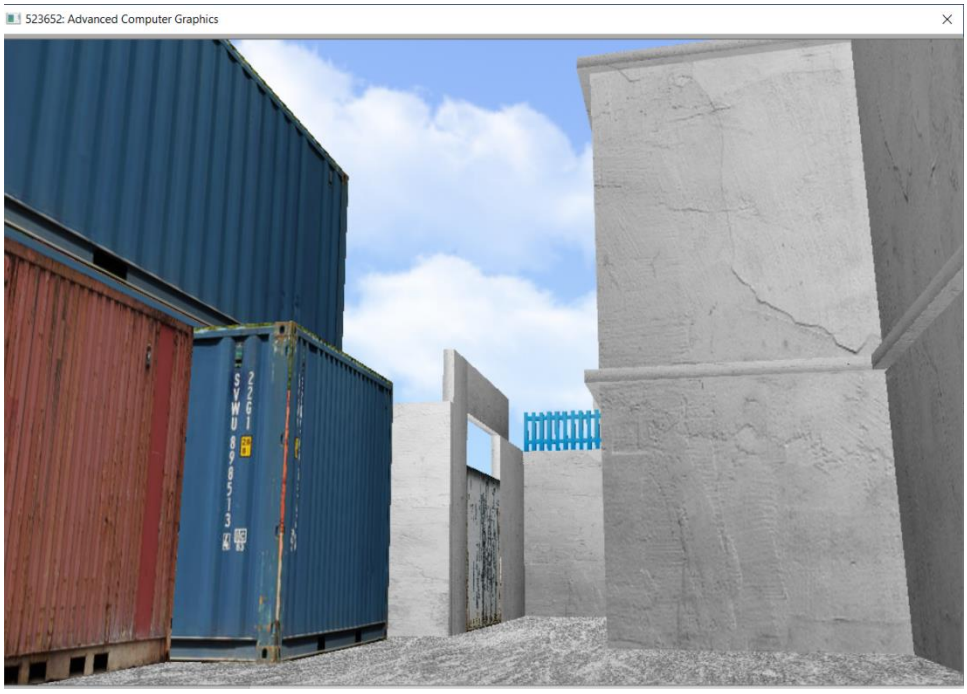
รูปหน้าจอผลการทำงาน




523652: Advanced Computer Graphics

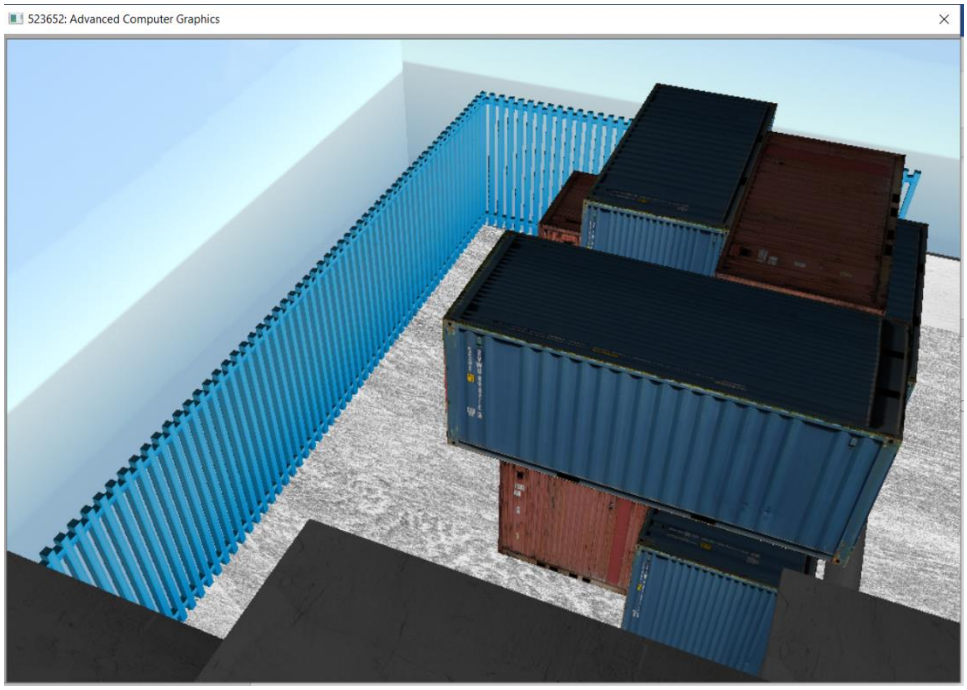

523652: Advanced Computer Graphics


523652: Advanced Computer Graphics


523652: Advanced Computer Graphics


523652: Advanced Computer Graphics


523652: Advanced Computer Graphics


523652: Advanced Computer Graphics


523652: Advanced Computer Graphics