Лабораторна робота № 3

Прозорість об’єктів, змішування кольорів та використання текстур на етапах формування реалістичного зображення.

Мета роботи: *Набути практичних навичок побудови реалістичного тривимірного зображення шляхом визначення та використання прозорості об’єктів, змішування кольорів та використання текстур.*

Завдання до роботи: *Модифікувати програму створену на попередній роботі додавши властивості прозорості до створених об’єктів, змішування кольорів та використавши технологію накладання текстури на графічні об’єкти.*

Теоретичні відомості.

Визначення прозорості графічних об’єктів.

В програмах, що генерують зображення за допомогою бібліотеки OpenGL колір можна задавати двома способами: явною вказівкою значень RGBA або вказівкою індексу кольору в палітрі. З прозорістю можна працювати тільки в режимі RGBA, тобто вказавши конкретні значення RGB і прозорості (альфа). При значенні альфа рівному 1 фрагмент вважається цілком непрозорим, а при 0 - цілком прозорим.

Тест на прозорість дозволяється і забороняється як і все інше командами glEnable і glDisable з параметром GL_ALPHA_TEST. У випадку, якщо тест не дозволений, вважається, що він пройшов успішно.

Процедура glAlphaFunc(func, ref) вказує бібліотеці, яким чином проводити перевірку фрагменту зображення на тест прозорості. Параметр ref - значення з діапазону [0, 1], з яким порівнюється значення, що надходить, альфа, а параметр func задає функцію порівняння значень альфа:

GL_NEVER - тест завжди завершується невдало;

GL_EQUAL - тест завершується позитивно, якщо значення, що надходить, дорівнює ref;

GL_NOTEQUAL - тест завершується позитивно, якщо значення, що надходить, не дорівнює ref;

GL_LESS - тест завершується позитивно, якщо значення, що надходить, менше ніж ref;

GL_LEQUAL - тест завершується позитивно, якщо значення, що надходить, менше або дорівнює ref;

GL_GREATER - тест завершується позитивно, якщо значення, що надходить, більше чим ref;

GL_GEQUAL - тест завершується позитивно, якщо значення, що надходить, більше або дорівнює ref;

GL_ALWAYS - тест завжди завершується позитивно.

Якщо тест завершився вдало, то фрагмент буде включено до зображення (якщо проходить всі інші тести), у протилежному випадку в буфері кадру на місці розташування даного фрагменту не відбувається ніяких змін.

Змішування кольорів.

Змішування кольорів - це комбінування значень кольору, що надходять до буферу кадру, у форматі RGBA з текстурного процессору і значень кольору, що зберігаються в буфері кадру. Правило, за яким OpenGL буде комбінувати кольори визначає процедура glBlendFunc(sfactor, dfactor). Параметр sfactor визначає вплив значень кольорів, що надходять з процесора на результуючий колір. А параметр dfactor задає вплив значень кольору, що зберігаються безпосередньо в буфері кадру. Усі можливі методи змішання кольорів описані нижче, де Rs, Gs, Bs, As - колір що надходить до буферу кадрів, а Rd, Gd, Bd, Ad - колір що зберігається в буфері кадрів. (Складові кольорів приймають значення в діапазоні від 0 до 1).

Параметр	**(*f*(*R*), *f*(*G*), *f*(*B*), *f*(*A*))**
GL_ZERO	(0, 0, 0, 0)
GL_ONE	(1, 1, 1, 1)
GL_SRC_COLOR	$(R_S / k_R, G_S / k_G, B_S / k_B, A_S / k_A)$
GL_ONE_MINUS_SRC_COLOR	$(1, 1, 1, 1) – (R_S / k_R, G_S / k_G, B_S / k_B, A_S / k_A)$
GL_DST_COLOR	$(R_d / k_R, G_d / k_G, B_d / k_B, A_d / k_A)$
GL_ONE_MINUS_DST_COLOR	(1, 1, 1, 1)

GL_SRC_ALPHA	$(R_d / k_R, G_d / k_G, B_d / k_B, A_d / k_A)$ – $(A_S / k_A, A_S / k_A, A_S / k_A, A_S / k_A)$
GL_ONE_MINUS_SRC_ALPHA	$(1, 1, 1, 1) – (A_S / k_A, A_S / k_A, A_S / k_A, A_S / k_A)$
GL_DST_ALPHA	$(A_D / k_A, A_D / k_A, A_D / k_A, A_D / k_A)$
GL_ONE_MINUS_DST_ALPHA	$(1, 1, 1, 1) – (A_D / k_A, A_D / k_A, A_D / k_A, A_D / k_A)$
GL_SRC_ALPHA_SATURATE	$(i, i, i, 1)$

Створення текстури об'єктів.

Формування зображення шляхом побудови об'єктів, що складаються з основних графічних примітивів не завжди достатньо для створення повноцінних та реалістичних тривимірних сцен. Не рідко виникає потреба накладати зображення на тривимірні об'єкти та повертати або масштабу вати зображення. Для вирішення даної проблеми існують текстури, які надають можливість покривати весь графічний об'єкт у вигляді мозаїки. Наприклад при створенні зображення підлоги або стіни будинку, не потрібно завантажувати зображення цегельної стіни або дерев'яного полу, а достатньо завантажити зображення одної дошки або цеглини та вказати, напрямок розмноження по всій площі. Розглянемо створення та накладення текстур на площину або на графічні об'єкти.

Підготовка растрового зображення для використання в якості текстури OpenGL. Після читання зображення з файлу необхідно підготувати його до використання в якості текстури: сюди входять два етапи - завдання формату зберігання зображення в пам'яті і коригування розмірів. Вказати OpenGL формат зберігання зображення можна командою glPixelStore [if] (GLenum pname; GLtype param); Також може знадобитися зміна формату зображення, типовим для. Bmp-файлів є зберігання компонентів кольору у вигляді BGR, для OpenGL потрібно перетворити в RGB. OpenGL вимагає, щоб розміри текстури були кратні ступені двійки. Після завершення підготовки образу можна створювати текстуру командами: glTexImage1D(GLenum target, GLint level, GLint components, GLsizei width, GLint border, GLenum format, GLenum type, void* pixels); для одновимірної текстури, під одномірної текстурою розуміється текстура висотою в 1 піксель, і glTexImage2D(GLenum target, GLint level, GLint components, GLsizei

width, GLsizei height, GLint border, GLenum format, GLenum type, void* pixels); для двовимірної текстури.

Текстурування – це процес накладання зображення на поверхню об’єкта. При накладанні зображення може змінювати масштаб, може відбувається зміна деталізації і т.д. Правила виконання таких перетворень визначаються параметрами текстури.

Для використання текстури необхідно спочатку завантажити в пам'ять потрібне зображення і передати його OpenGL.

Зчитування графічних даних з файлу та перетворення їх можна проводити вручну. Приклад коду для завантаження зображення дивись у додатку №1.

Накладання текстури на об’єкт.

При накладенні текстури треба враховувати випадок, коли розміри текстури відрізняються від віконних розмірів об'єкта, на який вона накладається. При цьому можливо як розтяг, так і стиснення зображення, і те, як будуть проводитися ці перетворення, може серйозно вплинути на якість побудованого зображення. Для визначення положення точки на текстурі використовується параметрична система координат (s, t), причому значення s і t перебувають у відрізку [0,1] (див. малюнок)

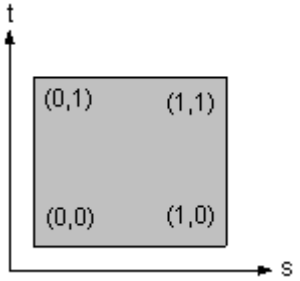

Рис. 8. Текстурні координати

Головним моментом при нанесенні зображення на об’єкт являється відображення, це є перетворення систем координат. Перед виконанням перетворення потрібно приписати кожній вершині відповідні точки в координатах текстури. В OpenGL це можна зробити кількома способами. Перший полягає в явному визначенні координат текстури для вершин об’єкта і виконується за допомогою команд:

```
glTexCoord[1 2 3 4][s i f d] (coord : GLtype);
glTexCoord[1 2 3 4][s i f d]v (coord : *GLtype);
```

Наприклад:

```
glTexCoord2f(0, 0);
glTexCoord2f(0, 1);
```

```
glTexCoord2f(1, 1);
glTexCoord2f(1, 0);
```

Другий спосіб полягає в визначенні деякої функції, яка вираховує координати текстури для кожної вершини. Для цього існують команди:

```
void glTexParameter[if](GLenum target, GLenum pname,  GLenum param);
void glTexParameter[if]v(GLenum target, GLenum pname, GLenum *params);
```

При цьому target може приймати значення GL_TEXTURE_1D або GL_TEXTURE_2D, pname визначає, яку властивість будемо міняти, а за допомогою param або params встановлюється нове значення. Можливі значення pname:

GL_TEXTURE_MAG_FILTER параметр param визначає функцію, яка буде використовуватися для збільшення (розтягування) текстури. При значенні GL_NEAREST буде використовуватися один (найближчий), а при значенні GL_LINEAR чотири найближчих елемента текстури. Значення за замовчуванням: GL_LINEAR .

GL_TEXTURE_WRAP_S параметр param встановлює значення координати s, якщо воно не входить у відрізок [0,1]. При значенні GL_ REPEAT ціла частина s відкидається, і в результаті зображення розмножується по поверхні. При значенні GL_CLAMP використовуються крайові значення: 0 або 1, що зручно використовувати, якщо на об'єкт накладається один образ. Значення за замовчуванням: GL_REPEAT .

GL_TEXTURE_WRAP_T аналогічно попереднього значення, тільки для координати t .

Використання режиму GL_NEAREST підвищує швидкість накладення текстури, однак при цьому знижується якість, так як на відміну від GL_LINEAR інтерполяція не проводиться.

Для того, щоб визначити як текстура буде взаємодіяти з матеріалом, з якого зроблений об'єкт, використовуються команди:

```
void glTexEnv[if](GLenum target, GLenum pname, GLtype param);
void glTexEnv[if]v(GLenum target, GLenum pname, GLtype * params);
```

Параметр target повинен бути рівний GL_TEXTURE_ENV, а в якості pname розглянемо тільки одне значення GL_TEXTURE_ENV_MODE, яке найбільш часто застосовується.

Найбільш часто використовувані значення параметра param: GL_MODULATE кінцевий колір знаходиться як добуток кольору точки на поверхні і кольору відповідної їй точки на текстурі.

GL_REPLACE в якості кінцевого кольору використовується колір точки на текстурі.

Можливі комбінації параметрів команди glTexEnv

GL_TEXTURE_WRAP_S	GL_CLAMP, GL_REPEAT, GL_CLAMP_TO_EDGE
GL_TEXTURE_WRAP_T	GL_CLAMP, GL_REPEAT, GL_CLAMP_TO_EDGE
GL_TEXTURE_WRAP_R	GL_CLAMP, GL_REPEAT, GL_CLAMP_TO_EDGE
GL_TEXTURE_MAG_FILTER	GL_NEAREST, GL_LINEAR
GL_TEXTURE_MIN_FILTER	GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR,GL_LINEAR_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_LINEAR

Щоб задати, які координати генерувати використовуються виклики:

```
glEnable(GL_TEXTURE_GEN_Q);
glDisable(GL_TEXTURE_GEN_Q);
glEnable(GL_TEXTURE_GEN_R);
glDisable(GL_TEXTURE_GEN_R);
glEnable(GL_TEXTURE_GEN_S);
glDisable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
glDisable(GL_TEXTURE_GEN_T);
```

Для quadric-об’єктів дозвіл генерації координат текстури виконує команда:

```
gluQuadricTexture(GLUquadricObj qobj, GLboolean textureCoords);
```

Виконання текстурування потрібно дозволити командами для одновимірного випадку:

```
glEnable(GL_TEXTURE_1D);
```

для двовимірного:

```
glEnable(GL_TEXTURE_2D);
```

Заборонити текстурування можна викликавши відповідно:

```
glDisable(GL_TEXTURE_1D);
glDisable(GL_TEXTURE_2D);
```

Порядок виконання роботи.

1. Створити проект, що містить консольну програму Win32.
2. Загрузити до каталогу проекта власне зобаження у форматі “BMP”.
3. Побудувати квадрат чорного кольору на білому фоні.
4. На створене у пункті №3 зображення нанести текстуру.

5. Побудувати фігуру відповідно до варіанта, що подана в Додатку №2.
6. На створене у пункті №5 зображення нанести текстуру .
7. Побудувати трьохвимірну фігуру відповідно до варіанта, що подана в Додатку №2.
8. На створене у пункті №7 зображення нанести текстуру .

На оцінку задовільно потрібно виконати завдання 1-4, добре 5-6, відмінно 7-8 .

УВАГА:

Обов'язковими для виконання є пункти, які не помічено відмітками: для отримання оцінки задовільно, для отримання оцінки добре та для отримання оцінки відмінно.

Оцінка задовільно ставиться у випадку виконання всіх обов'язкових пунктів роботи та пункту, який має відмітку для отримання оцінки задовільно.

Оцінка добре ставиться у випадку виконання всіх обов'язкових пунктів роботи та пунктів, які мають відмітку для отримання оцінки задовільно та для отримання оцінки добре.

Оцінка відмінно ставиться у випадку виконання всіх обов'язкових пунктів роботи та пунктів, які мають відмітку для отримання оцінки задовільно, для отримання оцінки добре та для отримання оцінки відмінно.

```
class BitmapBits
{
public:
BitmapBits() { bits = NULL; memset( &bmp, 0, sizeof( BITMAP ) ); };
~BitmapBits() { clear(); };
bool load( const char *path )
{
      clear();
      HBITMAP hbitmap = (HBITMAP) ::LoadImage( NULL, path,
      IMAGE_BITMAP, 0, 0, LR_CREATEDIBSECTION | LR_LOADFROMFILE );
      if( !hbitmap ) return false;
      ::GetObject( hbitmap, sizeof( BITMAP ), &bmp );
      bits = new unsigned char[ bmp.bmHeight * bmp.bmWidthBytes ];
      HDC hdc = GetDC( NULL );
      BITMAPINFO bi = { { sizeof(BITMAPINFOHEADER), bmp.bmWidth, bmp.bmHeight,
bmp.bmPlanes, bmp.bmBitsPixel } };
::GetDIBits( hdc, hbitmap, 0, bmp.bmHeight, bits, &bi, DIB_RGB_COLORS );
      ReleaseDC( NULL, hdc );
      ::DeleteObject( hbitmap );
      return true;
}
void clear( void )
{
      if( bits ) { delete bits; bits = NULL; }
      memset( &bmp, 0, sizeof( BITMAP ) );
}
unsigned char* getBits( void ) { return bits; }
int getWidth( void ) { return bmp.bmWidth; }
int getHeight( void ) { return bmp.bmHeight; }
int getBitsPerPixel( void ) { return bmp.bmBitsPixel; }
int getBitsPerLine( void ) { return bmp.bmWidthBytes; }
protected:
      BITMAP bmp;
      unsigned char *bits;
};
```

Додаток №2

№ варіанта	Завдання		Колір	
	Двохвимірна фігура	**Трьохвимірна фігура**	**Фон**	**Фігура**
1.	круг	куля	F6FF4F	3033FF
2.		конус	FF3ADB	4DFF21
3.		чотирикутна піраміду	FF0000	00FF00
4.		куб	0000FF	FF6DBD
5.	трикутник	циліндр	59FF61	FFFFFF
6.		трикутна зрізана піраміда	00FF00	F6FF4F
7.		5-кутна призму	FF2D4D	70FFEE
8.		зрізаний конус	3FFF82	1616FF
9.	шестикутник	4-кутна зрізана піраміду	FFFFFF	FF0000
10.		3-кутна піраміду	FF38E1	47FFE9
11.		зрізаний конус	3033FF	F6FF4F
12.		конус	4DFF21	FF3ADB
13.	зірку	4-кутна піраміду	FF2D4D	FF0000
14.		5-кутна призму	FF6DBD	0000FF
15.		циліндр	1616FF	59FF61
16.		3-кутна зрізана піраміда	F6FF4F	00FF00
17.	ромб	куб	FF0000	00FF00
18.		куля	FFFFFF	000000
19.		4-кутна зрізана піраміда	70FFEE	FFFFFF
20.		3-кутна піраміда	47FFE9	FF38E1
21.	трапецію	куля	E4FF8C	D582FF
22.		конус	35B1FF	FFF728
23.		4-кутна піраміда	FF3FEF	3F3FFF
24.		куб	19FF7C	FF2885
25.	Паралелограм	циліндр	3033FF	00FF00
26.		3-кутна зрізана піраміда	0000FF	FF38E1
27.		5-кутна призма	FF3ADB	FFFFFF
28.		зрізаний конус	70FFEE	FF38E1