Additional Effects

عملي مشترك

- سوف نتحدث في هذه المحاضرة عن مجموعة من التأثيرات التي تضيف للمشهد واقعية أكبر وجمالية.

Camera

-عند التحريك الكاميرا باستخدام تابع gluLookAt() بشكل مباشر قد يحدث بعض الأخطاء بسبب ضياع مركز الإحداثيات عند القيام بعمليات التحريك والدوران، لذلك ولتسهيل الاستخدام ولإعطاء حرية أكبر للكاميرا نستخدم Class Camera.

خطوات الاستخدام:

1. إضافة Camera.h وCamera.cpp إلى Project.
2. كتابة #include "Camera.h" ضمن Source إلى Project .
3. كتابة GLUT_BUILDNG_LIB إلى:
   project ⟶ properties ⟶ C/C++ ⟶ preprocessor ⟶ preprocessor Definition
4. إنشاء reference مع Class Camera، مثلاً : Camera Mycamera;
5. إنشاء object من Class Camera، مثلاً: mycamera = new Camera();
6. في تابع DrawGLScene() بعد تابع gluLoadIdentity(); نضع mycamera.Render();
7. استخدام توابع الموجودة ضمن Class Camera

مثال:

```
if (keys['S'])     MyCamera.MoveForward(-0.9);
if (keys['W'])     MyCamera.MoveForward(0.9 );
if (keys['D'])     MyCamera.RotateY(-2);
if (keys['A'])     MyCamera.RotateY;(2);
if (keys[VK_RIGHT])        MyCamera.MoveRight(1);
if(keys[VK_LEFT])  MyCamera.MoveRight(-1);
```

Fog

-يسمح OpenGl بإضافة ضباب للمشاهد لإعطاء جمالية أكبر ويساعد في إخفاء بعض الكائنات ببطء من خلال تفعيل تأثير الضباب.

-يقوم OpenGl بمزج لون الضباب مع لون الأشكال الهندسية الواقعة ضمن مجال الضباب فيعطي مشهد يظهر وكأنه يحتوي على ضباب حقيقي يحاكي العالم الواقعي.

تطبيق الضباب:

1. الضباب أحد الميزات الموجودة في OpenGl، لتفعيل الضباب نستخدم glEnable(GL_FOG);
2. لتحكم بالضباب بشكل أكبر يمكننا ذلك عبر تابع glFog، حيث يأخذ هذا التابع وسيطين: الأول يحدد الخاصية المراد تعديلها والثاني يحدد القيمة الجديدة للخاصية.
3. لتحديد لون الضباب يمكننا ذلك عبر ;(glFogfv(GL_FOG_COLOR,GLFloat[4] color
حيث تحدد المصفوفة color لون الضباب ومقدار الشفافية.
4. لتحديد المسافة المراد تطبيق الضباب عليها نقوم بتحديد نقطة البداية ونقطة النهاية التي تحدد العمق الذي يمتد عليه الضباب (أي امتداد على المحور Z) عبر تابعين :

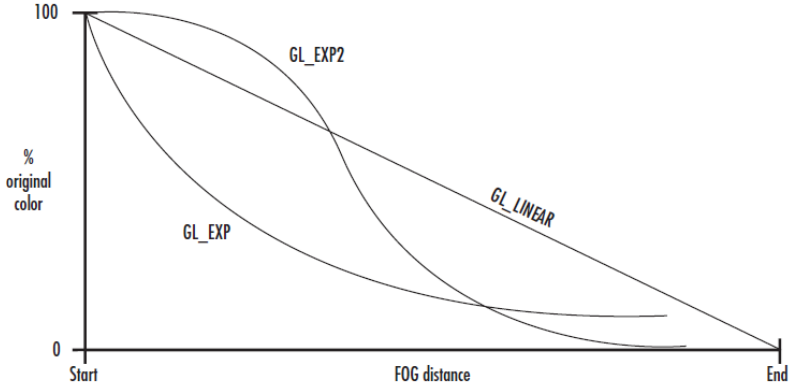
```
glFogf(GL_FOG_START,Z1);
glFogf(GL_FOG_END,Z2);
```

وبذلك سيمتد الضباب من Z1 إلى Z2.

5. يمكن تعين كثافة الضباب عبر تابع glFogf(GL_DENSITY,D);
حيث D قيمة الكثافة ويمكن أن تأخذ أي قيمة بين[1,0] .
6. يمكننا تحديد كيفية امتداد الضباب من البداية إلى النهاية وبإمكاننا أن نجعله كثيفاً في البداية و أقل كثافة في النهاية أو عكس وذلك عبر تابع glFogi(GL_Fog_MODE,Equation);

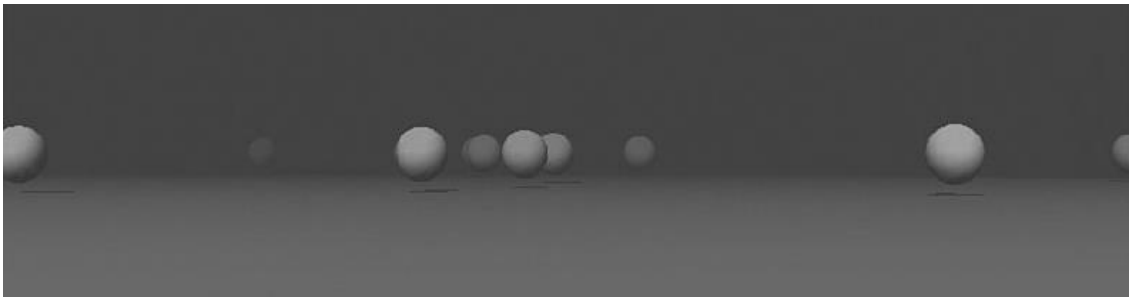
يمكن أن تكون Equation أي من القيم التالية :

Fog Mode	Fog Equation
GL_LINEAR	f = (end – c) / (end – start)
GL_EXP	f = exp(–d * c)
GL_EXP2	f = exp(–(d * c)²)



يوضح المخطط المجاور كيفية امتداد الضباب حسب المعادلات

وتوضح الصورة التالية تأثير الضباب في الـOpenGL:

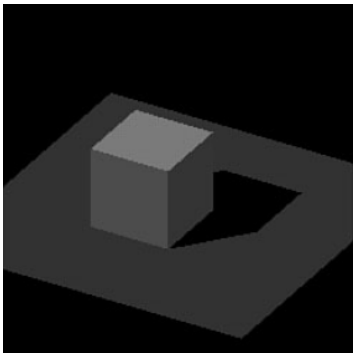


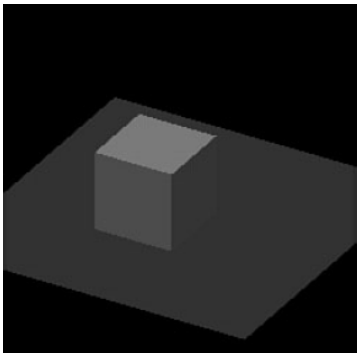
كل التوابع السابقة تُستدعى داخل الـ InitGL أو DrawGLScene

Shadow

بعد الحديث عن الألوان والإضاءة لابد من حديث عن الظلال التي تعطي المشاهد واقعية وفعالية بصرية.

في الصور التالية يظهر دور الظلال في إضافة الواقعية حيث كلا الشكلان تم تطبيق الإضاءة عليهم.





↙ يوجد نوعان أساسيان من الـGraphics بشكل عام:

Raster graphics

-تستخدم فكرة تلوين pixel أو إضاءة pixel، مثل OpenGl وهو لاينشأ ظلال للأشكال بشكل افتراضي لكنه أسرع من غيره.

Raytracing graphics

-يعتمد على فكرة تتبع أشعة الضوء المنعكسة من الكائنات لإظهار لون الأشكال وإعطاء لون لباقي الأشكال المحيطة والمتأثرة بالضوء.

-يوفر واقعية أكبر لكن بنفس الوقت يكون بطيئاً جداً ويحتوي على ظلال بشكل افتراضي للكائنات.

تطبيق الظلال:

لتكوين ظل لشكل معين نحتاج إلى معادلة السطح الذي يتكون عليه الظل ومنبع الضوء الذي يحدد اتجاه الظل وحجمه، ومنها نكون مصفوفة الظل والتي تكون إسقاط للشكل على السطح.

الخطوات

1. تضمين مكتبة math3d ضمن source.cpp التي تحتوي على التوابع اللازمة للظل.
2. تكوين معادلة السطح الذي سوف يتم عليه الإسقاط حيث نحتاج إلى 3 نقاط ليست على استقامة واحدة، ونحصل على معادلة بواسطة التابع

```
M3DVector4f equation;
M3DVector3f points[3] = {{ -9, -2, -8 }, { -9, -2, 8 }, { 10, -2, 8 }};
m3dGetPlaneEquation(equation, points[0], points[1], points[2]);
```

حيث equation تحتوي على معادلة السطح.

3. تكوين مصفوفة الظل، نحتاج إلى معادلة السطح ومنبع الضوء بالشكل

```
float Lightpos[] = { 0, 0, 0, 1.0 };
M3DMatrix44f shadow;
m3dMakePlanarShadowMatrix(shadow, equation, Lightpos);
```

يشكل التابع السابق مصفوفة الظل (shadow)، وتتم العمليات السابقة في التابع InitGL();

4. بعد رسم الشكل المراد صنع الظل له والإضاءة المناسبة له، نقوم بإيقاف تفعيل البعد الثالث

```
glDisable(GL_DEPTH_TEST)
```

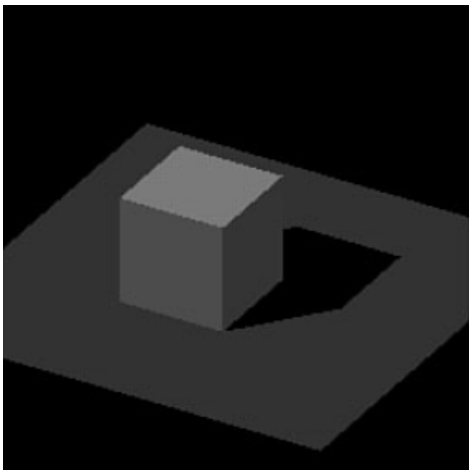
5. إيقاف تفعيل الإضاءة.
6. استخدام تابع glMaltMatrixf(); ليسمح بتطبيق الظل
7. إعطاء الظل لون أسود.
8. رسم الشكل المراد تطبيق الظل عليه مرة ثانية وعبر تابع glMaltMatrixf(); ويتم ضرب الشكل الثاني بمصفوفة الاسقاط فيعطي الظل المناسب للشكل
9. إعادة تفعيل البعد الثالث glEnable(GL_DEPTH_TEST).

برنامج إضافة الظل:

```
InitGl{
GLfloat lightPos[] = { -1.0f, 3.0f, 0.0f, 0.0f };
GLfloat LightAmb[] = {1.0f,1.0f,1.0f,1.0f};
GLfloat LightDiff[]  = {0.6f,0.6f,0.6f,1.0f};     // Transformation matrix to project shadow
M3DMatrix44f shadowMat;
 M3DVector3f points[3] = {{ -30.0f, -2.0f, -20.0f }, { -30.0f, -2.0f, 20.0f },{ 40.0f, -2.0f, 20.0f }};
}
glEnable(GL_LIGHTING); // Enable lighting
glLightfv(GL_LIGHT1,GL_POSITION,lightPos);
glLightfv(GL_LIGHT1,GL_AMBIENT,LightAmb);
glLightfv(GL_LIGHT1,GL_DIFFUSE,LightDiff);
glEnable(GL_LIGHT1);
M3DVector4f vPlaneEquation;
m3dGetPlaneEquation(vPlaneEquation, points[0], points[1], points[2]);
m3dMakePlanarShadowMatrix(shadowMat, vPlaneEquation, lightPos);
```

```
void DrawGLScene(GLvoid){
	gluLookAt(0,0,10,0,0,0,0,1,0);
	glBegin(GL_QUADS); //ground
		glColor3f(0,0.2,0);
		glVertex3f(40.0f, -2.0f, -40.0f); glVertex3f(-40.0f, -2.0f, -40.0f);
		glColor3f(0,1,0);
		glVertex3f(-40.0f, -2.0f, 40.0f); glVertex3f(40.0f, -2.0f, 40.0f);
	glEnd();
	glPushMatrix(); //light
		glTranslatef(lightPos[0],lightPos[1], lightPos[2]);
		glColor3ub(255,255,0);
		auxSolidSphere(0.1f);
	glPopMatrix();

	glPushMatrix(); //the original shape
		glEnable(GL_LIGHTING);
		glRotatef(10, 1.0f, 0.0f, 0.0f);
		glRotatef(45, 0.0f, 1.0f, 0.0f);
		glColor3f(1,0,0);
		auxSolidSphere(0.5);
	glPopMatrix();
	glDisable(GL_DEPTH_TEST);
	glDisable(GL_LIGHTING);
	glPushMatrix();  //the shadow
		glMultMatrixf((GLfloat *)shadowMat);
		glRotatef(10, 1.0f, 0.0f, 0.0f);
		glRotatef(45, 0.0f, 1.0f, 0.0f);
		glColor3f(0,0,0);
		auxSolidSphere(0.5);
	glPopMatrix();
	glEnable(GL_DEPTH_TEST);
}
```



Sound

-تمكننا Opengl من إضافة أصوات للمشهد مما يعطي واقعية وحيوية وجمالية أكبر.

كيفية تطبيق الصوت:

1. تنصيب برنامج openAL.exe
2. إضافة المكتبات الخاصة بالصوت إلى المسار:
   C:\Program Files\Microsoft Visual Studio 9.0\VC\lib
3. إضافة ملف al إلى المسار:
   C:\Program Files\Microsoft Visual Studio 9.0\VC\include
4. إضافة alut.lib ,OPENAL32.lib إلى:
   Project → Properties → Linker → input → Additional Dependencies
5. إضافة ملفات sound.cpp ,sound.h الى project.
6. كتابة "#include"sound.h في ملف source.cpp .
7. حتى نتمكن من تشغيل البرنامج openAL ونتحكم به نقوم بأخذ Object من INIT كما يلي:
   INIT initialize=INIT()
8. أخذ object من sound : sound sound1;
9. ضمن تابع InitGL() نقوم بتشغيل البرنامج عبر تابع : initialize.InitOpenAL();
10. لتحميل الصوت يجب أن يكون لاحقته .wav ويتم تحميله عبر sound1=sound("sound.wav") ضمن تابع InitGL()
11. يمكننا تشغيل الصوت عبر sound.play(); وإيقافه عبر sound.stop();

Blending & Transparency

-من أهم العناصر الأساسية لإضافة جمالية وواقعية للمشاهد هي الشفافية والأجسام العاكسة.

كل pixel في OpenGl يملك RGBA color يحدد RGB لون هذا الـ pixel وعنصر (Alpha) يحدد مقدار عتامة الـ pixel.

ففي حال كان جسم عاتم ولايحتوي شفافية ولايظهر الأجسام الموجودة خلفه تكون قيمة Alpha مساوية للصفر.

أما في حال كان الجسم شفاف بشكل كبير يمرر الضوء وتظهر جميع الأجسام خلفه تكون قيمة Alpha مساوية للواحد.

ويمكن وضع قيمة بين 0 و 1 للتدرج بالشفافية.

لإعطاء مظهر جسم شفاف في OpenGl يتم الاعتماد من مبدأ مزج الألوان Blending، حيث يتم مزج لون جسم شفاف مع لون الأجسام الواقعة خلفه عبر معادلة رياضية معينة فتظهر الأجسام الواقع من خلال جسم الشفاف بلون المزيج بين لون جسم شفاف ولون هذه الأجسام.

تطبيق الشفافية بالBlending:

Blending هو أحد الخواص الموجودة في opengl نقوم بتفعيلها في المكان الذي نريد تطبيق المزج عليه بـ:

```
glEnable(GL_BLEND);
```

عندما يتم تمكين المزج، تستخدم قيمة Alpha لمزج قيمة اللون الجديدة لنقطة ضوئية معطاة مع لون النقطة الضوئية الموجودة مسبقاً والمخزنة في framebuffer.

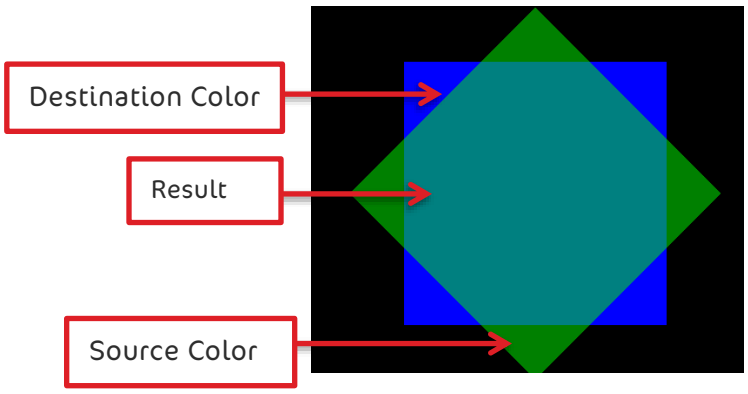
يتم التحكم بمقدار هذا المزج للوصول الى اللون المطلوب بـ:

```
glBLendFunc(GLenum  sfactor,GLenum  dfactor);
```

حيث:

Sfactor: يدل على كيفية حساب معامل مزج المصدر source (الجسم الشفاف الأمامي).

Dfactor: يدل على كيفية حساب معامل مزج الهدف destination (الجسم الواقع خلف الجسم الشفاف).



يمكن إن يأخذ dfactor,sfactor قيم متنوعة من جدول التالي مع العلم أن القيمة الافتراضية لـsfactor هي GL_ONE والقيمة الافتراضية لـ dfactor هي GL_ZERO وتعطي نفس النتيجة في حال عدم تفعيل المزج:

Function	RGB Blend Factors	Alpha Blend Factor
GL_ZERO	(0,0,0)	0
GL_ONE	(1,1,1)	1
GL_SRC_COLOR	(R_s,G_s,B_s)	A_s
GL_ONE_MINUS_SRC_COLOR	$(1,1,1) - (R_s,G_s,B_s)$	$1 - A_s$

GL_DST_COLOR	(R_d,G_d,B_d)	A_d
GL_ONE_MINUS_DST_COLOR	$(1,1,1) - (R_d,G_d,B_d)$	$1 - A_d$
GL_SRC_ALPHA	(A_s,A_s,A_s)	A_s
GL_ONE_MINUS_SRC_ALPHA	$(1,1,1) - (A_s,A_s,A_s)$	$1 - A_s$
GL_DST_ALPHA	(A_d,A_d,A_d)	A_d
GL_ONE_MINUS_DST_ALPHA	$(1,1,1) - (A_d,A_d,A_d)$	$1 - A_d$
GL_CONSTANT_COLOR	(R_c,G_c,B_c)	A_c
GL_ONE_MINUS_CONSTANT_COLOR	$(1,1,1) - (R_c,G_c,B_c)$	$1 - A_c$
GL_CONSTANT_ALPHA	(A_c,A_c,A_c)	A_c
GL_ONE_MINUS_CONSTANT_ALPHA	$(1,1,1) - (A_c,A_c,A_c)$	$1 - A_c$
GL_SRC_ALPHA_SATURATE	$(f,f,f)^*$	1

لكن للحصول على الشفافية المطلوبة نستخدم القيم التالية:

Sfactor= "GL_SRC_ALPHA"

Dfactor="GL_ONE_MINUS_SRC_ALPHA"

وتتم عملية المزج بإجراء المعادلة التالية:

Cf=(Cs*S)+(Cd*D)

حيث:

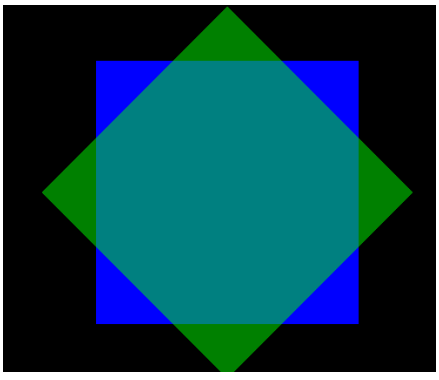
(final computed) Cf: **قيمة** RGBA **الممزوجة النهائية.**

(source color) Cs: **القيم اللونية للنقاط الضوئية الجديدة.**

(destination color) Cd: **القيم اللونية للنقاط الضوئية الموجودة مسبقاً.**

DgS: **عوامل مزج المصدر والهدف** (blending factors).

مثال:



```
glTranslatef(0,0,-20);
glColor3f(0,0,1);
glRectf(-5,5,5,-5);
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
glColor4f(0,1,0,0.5);
glRotatef(45,0,0,1);
glRectf(-5,5,5,-5);
```

↙ في حال كان لدينا صورة نريد إزالة الخلفية منها يمكننا ذلك عبر الـBlending

مثال:

لتنفيذ ذلك:

1. تحويل الصورة إلى(32 bit) tge. وذلك عبر Online Convert.
2. نضيف TgaLoader.h إلى Project.
3. كتابة "#include "TgaLoader.h ضمن source.cpp.
4. تعريف متحول من نوع TGAImage يخزن فيه عنوان الصورة في الذاكرة.
5. تفعيل texture وتحميل الصورة بواسطة التابع LoadTGA
6. رسم الشكل مع عمل الإكساء وتفعيل الشفافية.

مثال:

```
#include .....      // first of all
TGAImage  grass ;        // object for my tga image
initGL()
{ --------- // code
glEnable(GL_TEXTURE_2D);   // Enable Texture_2D
grass = LoadTGA("images//grass.tga"); // load tga image }
drawGLScene()   {
        glBindTexture(GL_TEXTURE_2D,grass.texID);
        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
        glBegin(GL_QUADS);
        glTexCoord2f(1,0);      glVertex3f(5,-5,-20);
        glTexCoord2f(1,1);      glVertex3f(5,5,-20);
        glTexCoord2f(0,1);      glVertex3f(-5,5,-20);
        glTexCoord2f(0,0);      glVertex3f(-5,-5,-20);
        glEnd();}
```

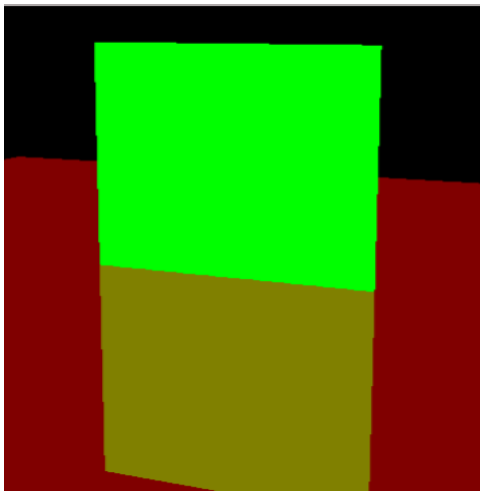
ملاحظات

الصورة يجب أن تكون (32bit) tga.

تأكد من كون اللون أبيض قبل تطبيق (1,1,1)textureglColorf.

Reflection

يمكننا إنشاء انعكاس لشكل أو لإعطاء مظهر وجود مرايا في المشهد عبر blending، حيث نقوم برسم الجسم أول مرة ثم نكرر رسمه مع إجراء انسحاب له ليظهر كأنه مقابل للجسم الأصلي، ثم نغير من شفافية السطح الموجود عليه الجسم الأصلي ليظهر بشكل سطح عاكس.



في OpenGl لا يوجد آلية مباشرة لتطبيق الانعكاس

إنما يتم إستخدام Blending.

نكون هُنا قد وصلنا الى نهاية القسم العملي من المقرر، كان معكم فريق البيانيات عملي، لا تنسونا من صالح دعائكم ❤