Computer Graphics

Prof. Jibum Kim

Department of Computer Science & Engineering

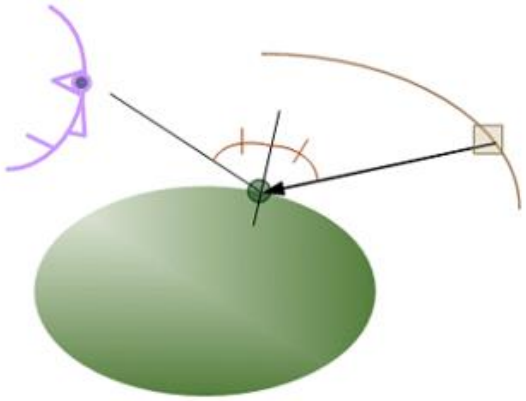
Incheon National University

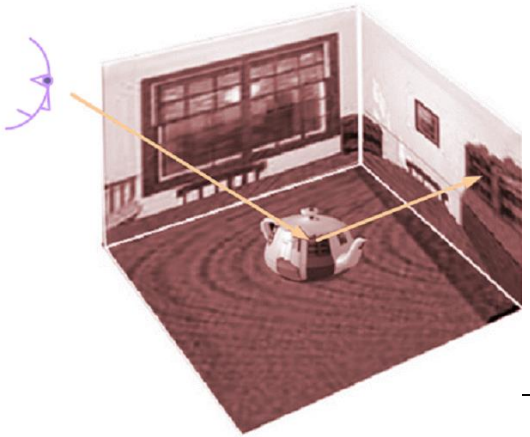
- **Environment Mapping (주변 매핑)**

- 달 표면에 착륙한 우주 비행사의 헬멧에는 주변의 분화구 모습이 반사되어 보인다
- Environment mapping은 이처럼 물체를 에워싼 주변 환경을 물체 면에 mapping 시키는 것을 말한다
- 즉, 물체 주변의 다른 물체의 모습이 texture로 사용된다

- Environment mapping이 가해지는 물체는 주로 표면이 매끄러운 물체로 조명 모델에서 보면 specular reflection (경면 반사)를 위주로 표현할 수 있는 물체를 말한다
- 이런 의미에서 environment mapping을 reflection mapping 이라고도 부른다
- Environment mapping 역시 앞에서 배운 2단계 mapping을 사용한다

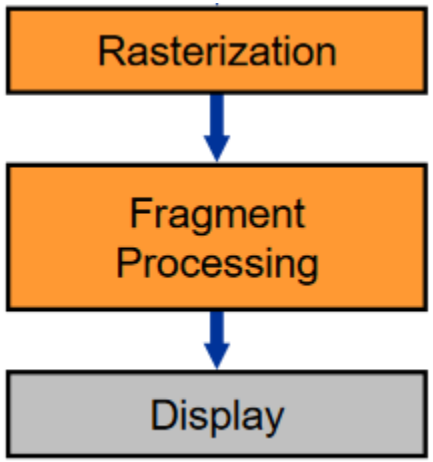
- 하지만, 2단계 (O-mapping)에서 아래와 같이 시점 반사 벡터 방식으로 구현된다
- 즉, O-mapping시에 Viewer에서 물체 면을 향한 벡터 (ray)가 정반사되어 벽면에 부딪치는 곳의 texture를 물체 면의 texture로 mapping 시킨다
- Viewer의 위치가 달라지면 반사광이 부딪치는 벽면의 위치가 달라지므로 environment mapping 결과도 달라진다
- 1단계의 중개면으로는 주로 cube가 많이 사용된다





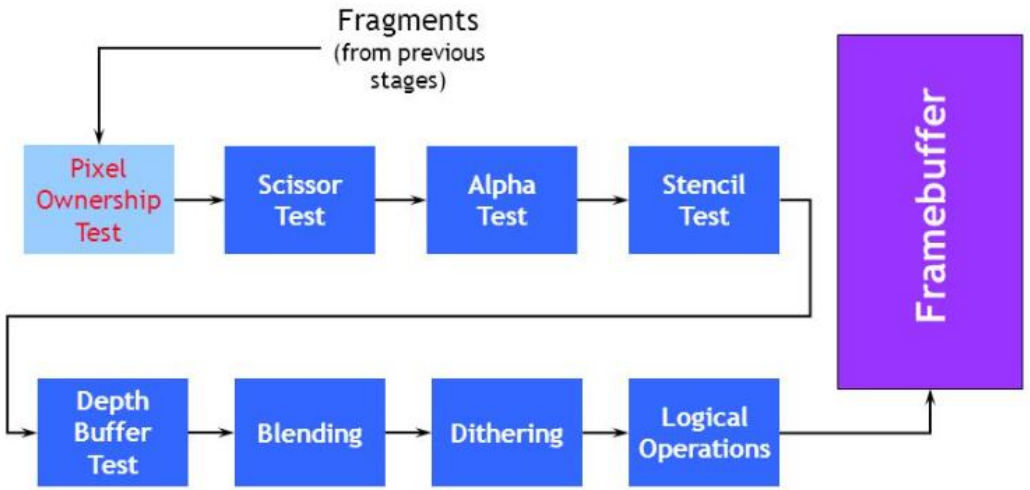
- **Fragment 연산**

- 화면 화소의 색은 Frame buffer에 값이 기록되는 순간에 결정 된다. Rasterization의 결과로 Frame buffer에 기록되기 이전 단계의 화소를 **Fragment (프래그먼트)**라 부른다.
- Fragment란 화소에 대응하는 추상적인 개념으로, 화소 위치를 비롯하여 해당 화소의 색, 깊이 (depth) , texture등, 하나의 화소를 채우기 위해 필요한 정보를 지닌 기본 단위이다

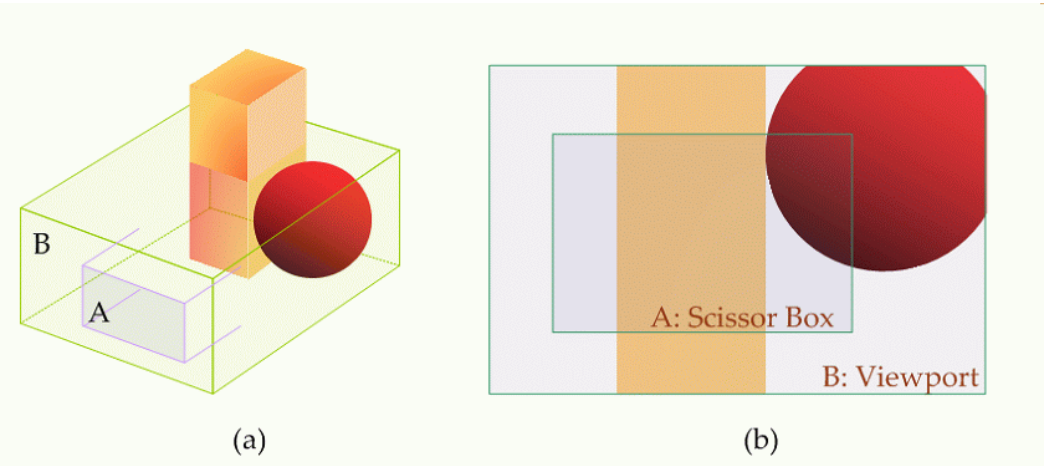


- Fragment가 화소로 변하여 frame buffer에 저장되기 전까지 일련의 fragment 연산이 가해진다
- 연산의 전반부는 테스트 작업으로, 주어진 테스트를 가할지 말지는 glEnable()과 glDisable() 함수에 의해 적용된다
- 모든 test (총 4개)를 통과한 fragment만 그림의 마지막 단계인 blending, dithering, 논리연산이 실행되고, 최종 결과가 버퍼에 기록된다

- OpenGL에는 fragment별로 총 4가지의 test를 거친다
- 1. Scissor test 2. alpha test 3. stencil test 4. depth buffer test
- 각 fragment는 최종적으로 frame buffer에 기록되기 전에 아래 순서와 같이 4가지 종류의 test를 통과해야만 한다. Test를 통과하지 못하면 frame buffer에 기록되지 않고 그 fragment는 보이지 않는다
- 이와 같은 작업을 masking 이라고도 한다



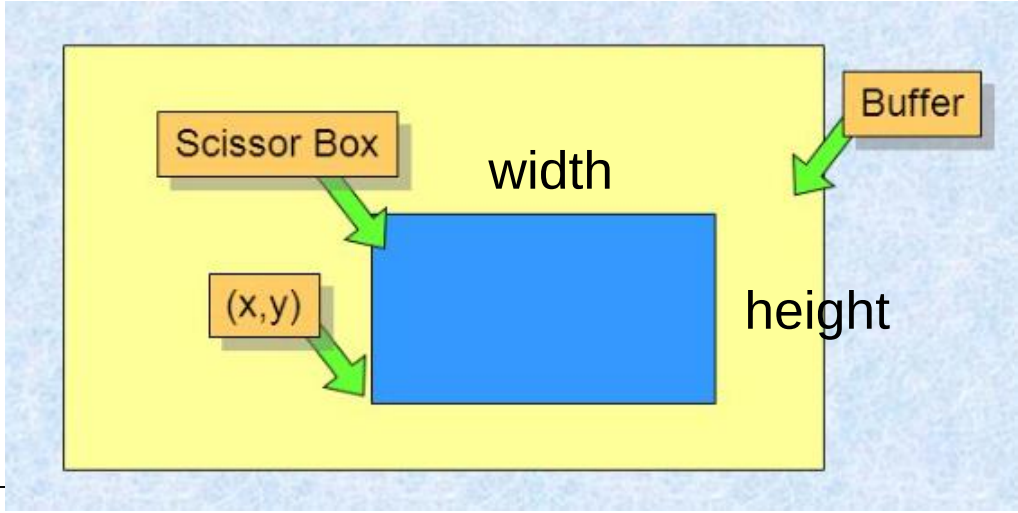
- **1. Scissor test (시저 테스트)**
- **아래 그림과 같이 Viewport 내에 추가로 scissor box (아래 그림 A)라는 직사각형 영역을 정하고 이 영역의 바깥에 있는 fragment는 clipping 시키는 테스트**
- **일종의 추가적인 clipping 효과로 특수한 효과를 줄 수 있다**



- **OpenGL에서의 사용법**

```
glEnable(GL_SCISSOR_TEST);
glScissor(x, y, width, height)
```

- **Viewport 좌표계로 mask의 왼쪽 아래로 부터 (x, y) pixel 만큼 떨어진 위치로 부터 가로 (width) 세로 (height) pixel 크기의 scissor box 생성**
- **이 scissor box의 바깥에 있는 fragment는 보이지 않는다**



- 다음 예제는 애니메이션 예제 이다
- 여기에 주석부분을 없애면 scissor test를 수행할 수 있다
- 왼쪽 아래 부분에 scissor box를 만들어서 이부분만 보이게 하였다
- `// glEnable(GL_SCISSOR_TEST);`
- `// glScissor(0.0, 0.0, 250.0, 250.0);`

- **https://www.dropbox.com/s/a888aaqunrnbtsp/fragment_1.txt?dl=0**

- **2. Alpha test**
- **Alpha값을 기준으로 해당 fragment를 제외 시키는 test**
- **Default는 GL_ALWAYS (항상 통과)**

```
glEnable(GL_ALPHA_TEST);
glAlphaFunc(func, ref);
```

- **예: glAlphaFunc(GL_LESS, 0.2)**
- **// alpha값이 0.2보다 작은 fragment는 통과**

void **glAlphaFunc**(GLenum func, GLclampf ref);	
GL_NEVER	GL_ALWAYS
GL_LESS	GL_LEQUAL
GL_GREATER	GL_GEQUAL
GL_NOTEQUAL	GL_EQUAL

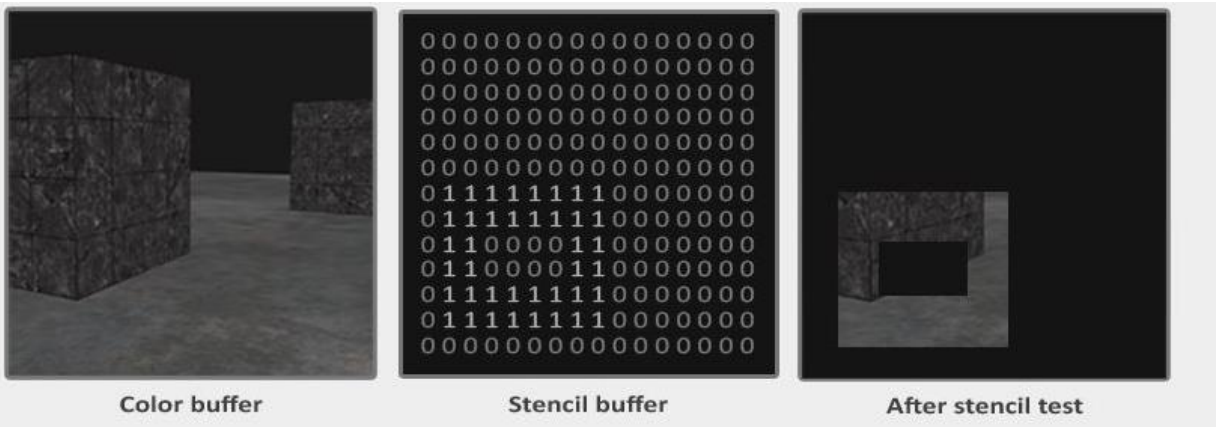
- 색을 나타낼 때에 RGB 대신에 RGBA를 사용함으로써 alpha값을 적용할 수 있다

- Alpha 값은 불 투명도를 나타내며 Alpha 값=1이면 완전히 불투명함을 alpha값=0이면 완전히 투명함을 말한다
- 예: glColor4f(R, G, B, A)

- 다음 예제에 있는 주석 부분을 제거하면 alpha test를 수행할 수 있다
- //glEnable(GL_ALPHA_TEST);
- //glAlphaFunc(GL_LESS, 0.5);

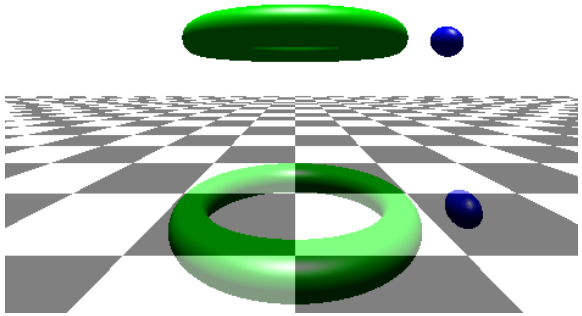
```
#include <cstdlib>
#include <cmath>
#include <GL/glut.h>


void display(void)
{
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glClear(GL_COLOR_BUFFER_BIT);

//glEnable(GL_ALPHA_TEST);
//glAlphaFunc(GL_LESS, 0.5);

glBegin (GL_TRIANGLES);
glColor4f(1.0, 0.0, 0.0, 0.7);
glVertex3f(0.1, 0.9, 0.0);
glVertex3f(0.1, 0.1, 0.0);
glVertex3f(0.7, 0.5, 0.0);
glEnd();

glBegin (GL_TRIANGLES);
glColor4f(0.0, 1.0, 0.0, 0.3);
glVertex3f(0.9, 0.9, 0.0);
glVertex3f(0.3, 0.5, 0.0);
glVertex3f(0.9, 0.1, 0.0);
glEnd();


glFlush();
}


int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize (700, 700);
glutCreateWindow (argv[0]);
glClearColor (1.0, 1.0, 1.0, 0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0, 1, 0, 1, -1, 1);
glutDisplayFunc (display);
glutMainLoop();
return 0;
}
```

- **3. Stencil test**
- **depth buffer test 바로 전에 적용하는 test로서 주로 특수효과를 줄 때에 사용한다**
- **Stencil buffer에 저장된 임의의 모양을 기준으로 시저 박스와 같이 masking. 시저 박스 보다 더 다양한 모양 (mask) 및 효과를 줄 수 있다**
- **스텐실 테스트에 의해 stencil buffer에 0으로 기록된 부분은 모두 제외되서 보이게 된다**



- Stencil test를 이용한 그림자 그리기 예
- 원래 물체를 Reflection을 적용하였고 stencil test를 이용하여 바닥 밖의 물체를 제거하였다

- 다음 예제는 reflection과 blending을 사용하여 반사 효과를 준 애니메이션 이다
- Space 키: 애니메이션 on/off
- Up/down 키: 애니메이션 스피드 업/다운



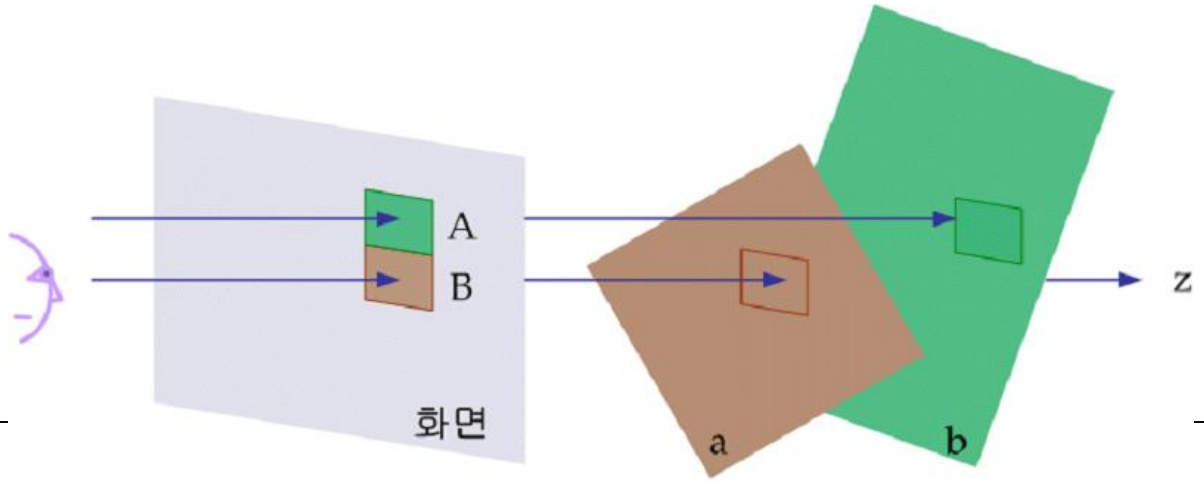
```
#include <iostream>
#include <GL/glut.h>

using namespace std;

// Globals.
static float latAngle = 0.0; // Latitudinal angle.
static float longAngle = 0.0; // Longitudinal angle.
static int isAnimate = 0; // Animated?
static int animationPeriod = 100; // Time interval between frames.

// Draw ball flying around torus.
void drawFlyingBallAndTorus(void)
{

  glPushMatrix();

  glTranslatef(0.0, 10.0, -15.0);
  glRotatef(90.0, 1.0, 0.0, 0.0);

  // Fixed torus.
  glColor4f(0.0, 1.0, 0.0, 1.0); // High alpha for opacity.
  glutSolidTorus(2.0, 12.0, 80, 80);

  // Begin revolving ball.
  glRotatef(longAngle, 0.0, 0.0, 1.0);

  glTranslatef(12.0, 0.0, 0.0);
  glRotatef(latAngle, 0.0, 1.0, 0.0);
  glTranslatef(-12.0, 0.0, 0.0);

  glTranslatef(20.0, 0.0, 0.0);

  glColor4f(0.0, 0.0, 1.0, 1.0); // High alpha for opacity.
  glutSolidSphere(2.0, 20, 20);
  // End revolving ball.

  glPopMatrix();
}

// Timer function.
void animate(int value)
{
  if (isAnimate)
  {
    latAngle += 5.0;
                    if (latAngle > 360.0) latAngle -= 360.0;
    longAngle += 1.0;
                    if (longAngle > 360.0) longAngle -= 360.0;

                    glutPostRedisplay();
    glutTimerFunc(animationPeriod, animate, 1);
  }
}

// Initialization routine.
void setup(void)
{
  glClearColor(1.0, 1.0, 1.0, 0.0);
  glEnable(GL_DEPTH_TEST); // Enable depth testing.

  // Turn on OpenGL lighting.
  glEnable(GL_LIGHTING);

  // Light property vectors.
  float lightAmb[] = { 0.0, 0.0, 0.0, 1.0 };
  float lightDifAndSpec[] = { 1.0, 1.0, 1.0, 1.0 };
  float lightPos[] = {0.0, 1.0, 0.0, 0.0 };
  float globalAmb[] = { 0.2, 0.2, 0.2, 1.0 };

  // Light properties.
  glLightfv(GL_LIGHT0, GL_AMBIENT, lightAmb);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, lightDifAndSpec);
  glLightfv(GL_LIGHT0, GL_SPECULAR, lightDifAndSpec);
  glLightfv(GL_LIGHT0, GL_POSITION, lightPos);

  glEnable(GL_LIGHT0); // Enable particular light source.
```

- 앞의 코드를 stencil buffer와 stencil test를 이용하여Disc 모양의 영역에만 반사효과를 주도록 코드를 변경하였다
- Stencil buffer를 사용하면 이와 같이 특정 영역에만 (이 경우 disc 모양) 특정한 효과를 줄 수 있다
- Space 키: 애니메이션 on/off
- 방향 키: disc 모양의 영역 움직임

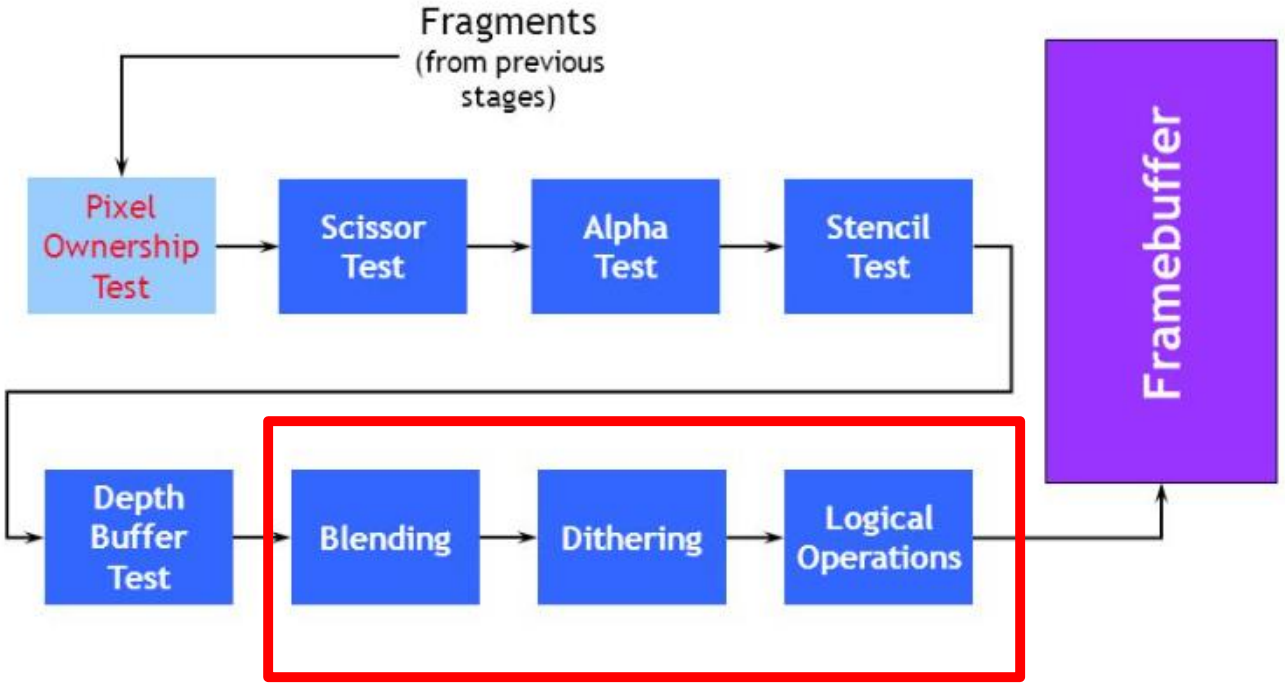
```
#include <iostream>
#include <cmath>
#include <GL/glut.h>

#define PI 3.1415926535897932384

using namespace std;

// Globals.
static float latAngle = 0.0; // Latitudinal angle.
static float longAngle = 0.0; // Longitudinal angle.
static int isAnimate = 0; // Animated?
static int animationPeriod = 75; // Time interval between frames.
static float xVal = 0.0, zVal = 12.0; // Displacement parameters of disc.

// Draw ball flying around torus.
void drawFlyingBallAndTorus(void)
{

  glPushMatrix();

  glTranslatef(0.0, 10.0, -15.0);
  glRotatef(90.0, 1.0, 0.0, 0.0);

  // Fixed torus.
  glColor4f(0.0, 1.0, 0.0, 1.0); // High alpha for opacity.
  glutSolidTorus(2.0, 12.0, 80, 80);

    // Begin revolving ball.
  glRotatef(longAngle, 0.0, 0.0, 1.0);

  glTranslatef(12.0, 0.0, 0.0);
  glRotatef(latAngle, 0.0, 1.0, 0.0);
  glTranslatef(-12.0, 0.0, 0.0);

  glTranslatef(20.0, 0.0, 0.0);

  glColor4f(0.0, 0.0, 1.0, 1.0); // High alpha for opacity.
  glutSolidSphere(2.0, 20, 20);
  // End revolving ball.

  glPopMatrix();
}

// Draw reflective disc centered at the input position.
void drawReflectiveDisc(float x, float z)
{
  glPushMatrix();
  glTranslatef(x, 0.0, z);
  glColor4f(0.0, 0.0, 0.0, 0.5); // Low alpha for blending.
  glNormal3f(0.0, 1.0, 0.0);
  glBegin(GL_POLYGON);
    for(int i = 0; i < 100; ++i)
                                {
      float t = 2*PI*i/100;
      glVertex3f(7.5*sin(t), 0.0, 7.5*cos(t));
                                }
  glEnd();
  glPopMatrix();
}

// Timer function.
void animate(int value)
{
  if (isAnimate)
  {
    latAngle += 5.0;
                                if (latAngle > 360.0) latAngle -= 360.0;
    longAngle += 1.0;
                                if (longAngle > 360.0) longAngle -= 360.0;
                                glutPostRedisplay();
    glutTimerFunc(animationPeriod, animate, 1);
  }
}
// Initialization routine.
```

- **4. Depth buffer test**
- 깊이 버퍼 테스트란 시점에 가까운 fragment가 멀리 있는 fragment를 가려 가까운 fragment의 깊이가 버퍼에 기록되는 것을 말한다
- 이는 주로 은면 제거 작업에 사용된다
- 이러한 기본 논리를 적용하려면 glEnable(GL_DEPTH_TEST) 함수를 호출해야 한다

- Depth buffer test에는 아래 그림과 같이 평행한 여러 개의 ray가 viewer로 부터 발사되어 나가는 것으로 가정한다
- 각 ray마다 서로 다른 화소를 통과하므로 광선의 수는 화면의 화소 수와 동일하다
- Viewer로부터 나가는 ray가 만나는 첫 물체 면의 색 (즉, viewer와 가장 가까운 물체 면의 색)이 화소의 색이 된다
- Z-buffer (depth buffer)란 모든 개별 화소에 대해 현재 ray와 교차된 물체의 깊이 정보를 저장하고 있는 메모리를 말한다

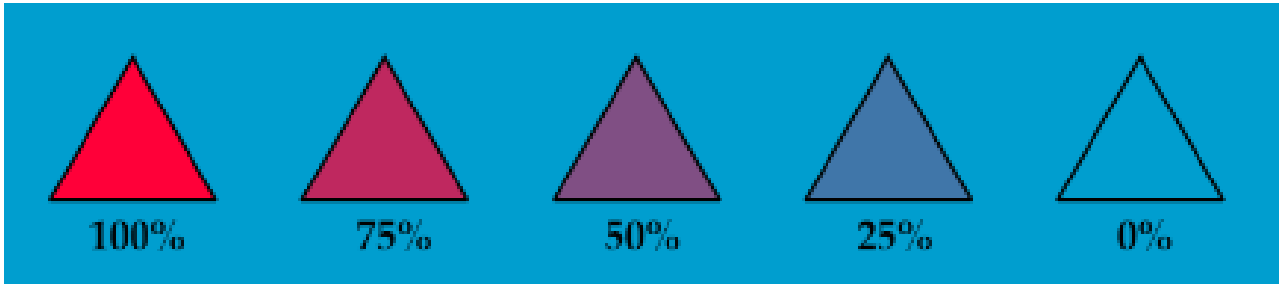


Fragment 연산에서 4개의 test를 모두 통과하면 마지막 단계인 blending, dithering, logical operations가 실행된다

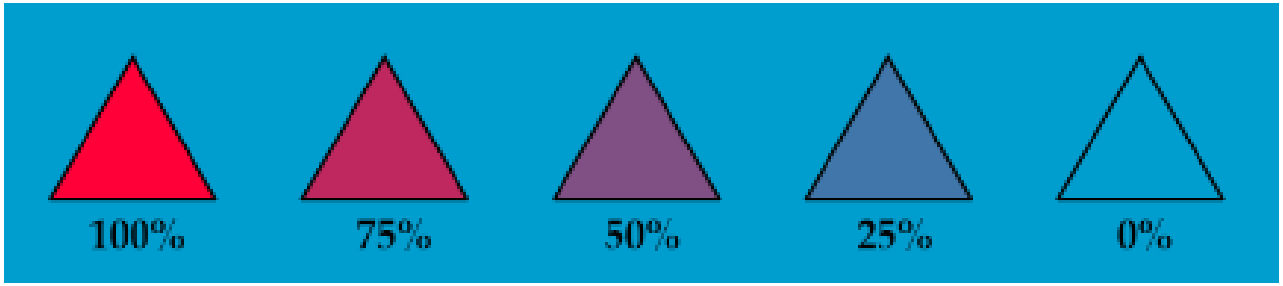


- **Blending**

- 아래 그림과 같이 원래 배경이 파란색이고 새로운 물체 (삼각형)의 색이 빨간색이라고 하자
- 이 때 새로운 물체가 투명하다 (transparent)의 의미는 ?
- 물체 뒤의 색이 보인다. 불투명하다 (opaque)의 의미는?
- 투명도는 (R,G,B,A)에서 A, Alpha값이라 하고 [0, 1]사이 값을 갖는다
- Alpha값이 1 (100%)에 가까울 수록 불투명하다고 한다
- 반대로 Alpha값이 0 (0%)이면 완전히 투명한 것이다



- 그림은 새로운 물체 (빨간색) 의 투명도 (Alpha 값)에 따른 결과이다
- Alpha=1.0 (100% 불투명): 최종색=1.0*새로운 물체의 색 + 0.0*원래 색
- Alpha=0.5 (50% 불투명): 최종색=0.5*새로운 물체의 색 + 0.5*원래 색
- Alpha=0.0 (0% 불투명): 최종색=0.0*새로운 물체의 색+ 1.0*원래 색



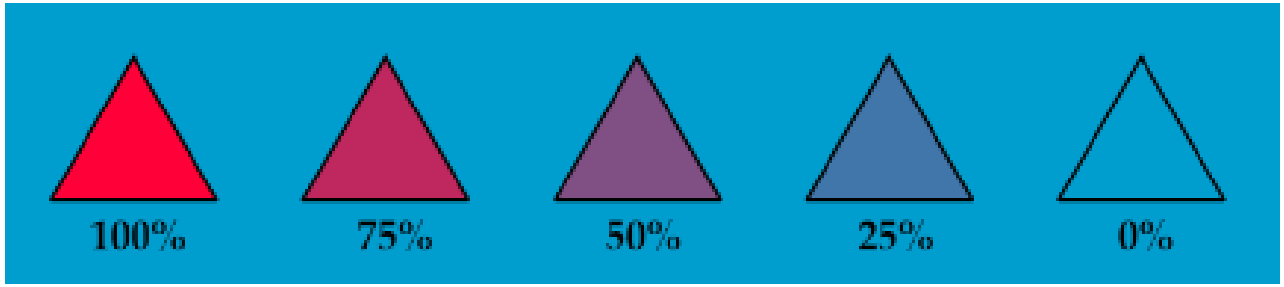
- 이 수식을 일반화 화면 다음과 같은 수식을 얻는다

최종색 = (새 물체 Alpha값) *새로운 물체의 색 + (1-새 물체 Alpha값) *원래 색

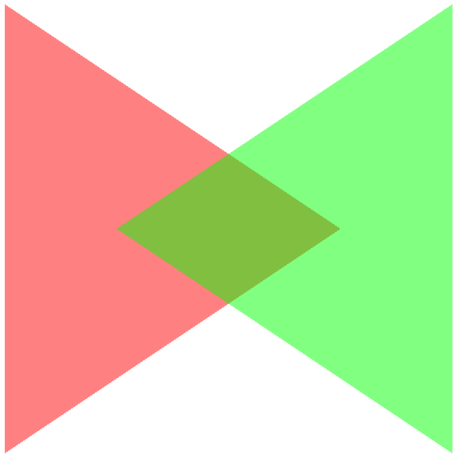
- **투명도를 이용한 기본적인 blending 수식**
- **Source: 새로운 물체의 색 (R,G,B)=(src_R, src_G, src_B)**
- **Destination: 원래 색 (R,G,B)=(dst_R, dst_G, dst_B)**
- **src_α : 새로운 물체의 투명도 ([0,1] 사이 값)**

최종색 = src_α *(새로운 물체의 색) + (1- src_α) * (원래색)

$$(R_f, G_f, B_f) = src_\alpha * (src_R, src_G, src_B) + (1 - src_\alpha) * (dst_R, dst_G, dst_B)$$



- 즉, 어떤 물체가 투명하게 보이기 위하여 최종 색을 정할 때 기존 물체의 색 (destination)과 새로운 물체의 색 (source)을 섞어서 (blending)보여 준다
- 두 물체의 색을 혼합하여 새로운 색을 만드는 것을 blending이라고 한다
- Destination과 source의 색이 섞는 비율을 blending factor라 한다
- 앞의 수식에서 source와 destination의 blending factor는 각각 얼마인가?
- Source의 blending factor: src_α
- Destination의 blending factor: $1 - src_\alpha$



- 예: 특정 pixel에서 현재 색의 (R,G,B)=(0.5, 0.5, 0.5)이고 새로운 물체의 색이 (R,G,B)=(1.0, 0.6, 0.4)이다. 새로운 물체의 alpha값이 0.5일 때 앞의 수식을 이용하여 이 pixel의 최종 색을 구해보자
- $(R_f, G_f, B_f) = src_\alpha * (src_R, src_G, src_B) + (1 - src_\alpha) * (dst_R, dst_G, dst_B)$

- **OpenGL에서의 blending 사용**

1. OpenGL에서 Blending enable 시킴

- glEnable(GL_BLEND);

2. Blending시 source와 destination의 blending factor 정함

glBlendFunc(source의 blending factor, destination의 blending factor)

예: source의 blending factor: src_α

destination의 blending factor: $1 - src_\alpha$

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

- 예: 투명도 및 blending 에 대한 코드
- 무엇이 source이고 무엇이 destination인지 파악해 보자
- 이 경우에는 destination이 배경색이 된다
- Source의 alpha값을 0.0에서 1.0까지 증가시켜 보자

Destination (원래 색)	glClearColor(0.0, 0.0, 1.0, 0.0);	파란색
Source (새로운 물체의 색)	glColor4f(1.0, 0.0, 0.0, 0.0);	빨간색
Source의 alpha값	0.0	완벽히 투명

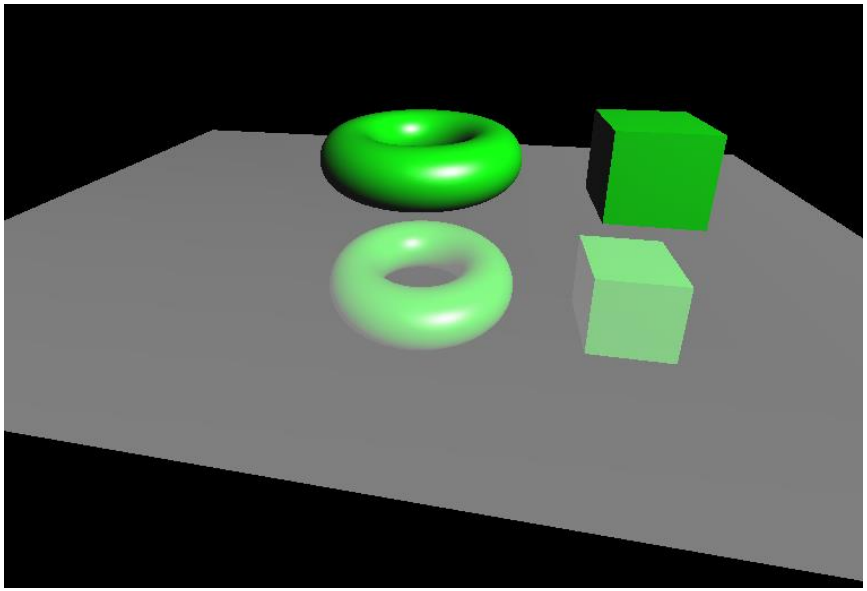
```
#include <iostream>
#include <GL/glut.h>

using namespace std;

// Globals.
static int isWire = 0; // Is wireframe?

// Drawing routine.
void drawScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT);

  if (isWire) glPolygonMode(GL_FRONT_AND_BACK, GL_LINE); else glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

  glEnable(GL_BLEND);
  glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

  // Draw square annulus.
  glBegin(GL_TRIANGLE_STRIP);
  glColor4f(1.0, 0.0, 0.0, 0.0);
  glVertex3f(10.0, 10.0, 0.0);
  glVertex3f(70.0, 10.0, 0.0);
  glVertex3f(30.0, 70.0, 0.0);
  glEnd();
  glFlush();
}

// Initialization routine.
void setup(void)
{
  glClearColor(0.0, 0.0, 1.0, 0.0);
}

// OpenGL window reshape routine.
void resize(int w, int h)
{
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(0.0, 100.0, 0.0, 100.0, -1.0, 1.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

// Keyboard input processing routine.
void keyInput(unsigned char key, int x, int y)
{
  switch(key)
  {
    case ' ':
      if (isWire == 0) isWire = 1;
      else isWire = 0;
      glutPostRedisplay();
      break;
    case 27:
      exit(0);
      break;
    default:
      break;
  }
}

// Routine to output interaction instructions to the C++ window.
void printInteraction(void)
{
  cout << "Interaction:" << endl;
  cout << "Press the space bar to toggle between wireframe and filled." << endl;
}

// Main routine.
int main(int argc, char **argv)
{
  printInteraction();
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
  glutInitWindowSize(500, 500);
```

- 이번에는 두 물체를 사용하여 투명도를 이용하여 blending 효과를 주었다
- 다음 코드를 실행해 보고 주석을 제거하여 실행하여 보자

```
#include <cstdlib>
#include <cmath>
#include <GL/glut.h>


void display(void)
{
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glClear(GL_COLOR_BUFFER_BIT);

glEnable (GL_BLEND);
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

glBegin (GL_TRIANGLES);
glColor4f(1.0, 0.0, 0.0, 0.5);
glVertex3f(0.1, 0.9, 0.0);
glVertex3f(0.1, 0.1, 0.0);
glVertex3f(0.7, 0.5, 0.0);
glEnd();

//glBegin (GL_TRIANGLES);
//glColor4f(0.0, 1.0, 0.0, 0.5);
//glVertex3f(0.9, 0.9, 0.0);
//glVertex3f(0.3, 0.5, 0.0);
//glVertex3f(0.9, 0.1, 0.0);
//glEnd();


glFlush();
}


int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize (700, 700);
glutCreateWindow (argv[0]);
glClearColor (1.0, 1.0, 1.0, 0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0, 1, 0, 1, -1, 1);
glutDisplayFunc (display);
glutMainLoop();
return 0;
}
```

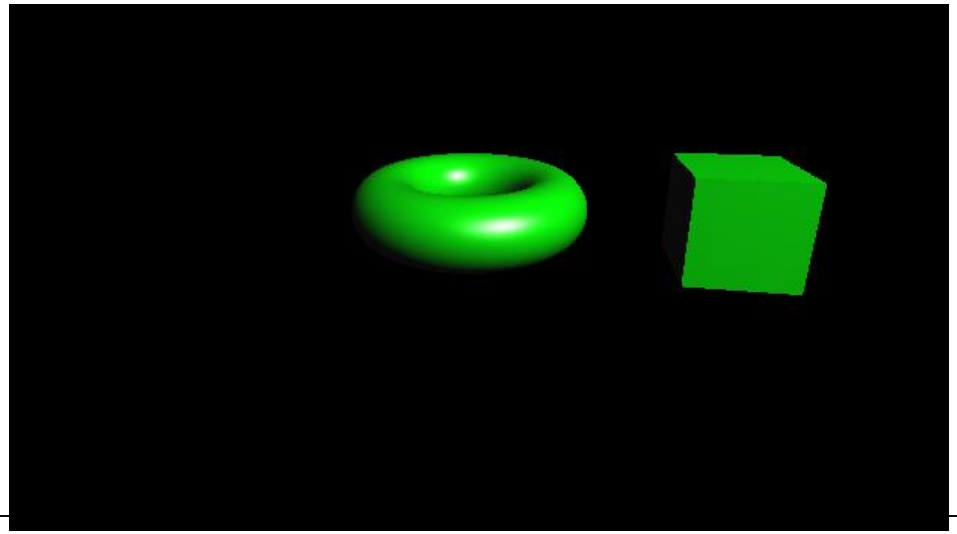
- **Blending을 이용한 torus 반사**

- 다음과 같이 blending 효과를 이용하여 물체가 비치는 것처럼 보이게 하기 위하여 다음과 같이 수행하였다



- 1. Glut objects 2개 (torus, cube) 생성 및 이동
- Lighting 효과 (빛 색 및 반사율 조절) 추가

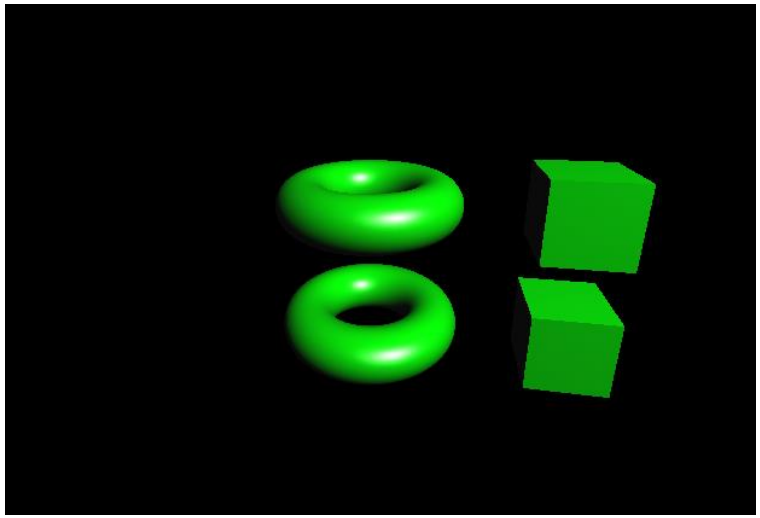
```
gluLookAt(5.0, 10.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
glRotatef(90,1,0,0);
glutSolidTorus(2.0, 5.0, 80, 80);
glTranslatef(15.0, 0.0, 0.0);
glutSolidCube(6.0);
```



```
#include <iostream>

#include<GL/glut.h>

using namespace std;

void setup(void)
{
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glEnable(GL_DEPTH_TEST); // Enable depth testing.

   // Turn on OpenGL lighting.
   glEnable(GL_LIGHTING);

   // Light property vectors.
   float lightAmb[]  = { 0.0, 0.0, 0.0, 1.0 };
   float lightDifAndSpec[] = { 1.0, 1.0, 1.0, 1.0 };
   float lightPos[] = {30.0, 30.0, 30.0, 1.0 };

   // Light properties.
   glLightfv(GL_LIGHT0,  GL_AMBIENT, lightAmb);
   glLightfv(GL_LIGHT0,  GL_DIFFUSE, lightDifAndSpec);
   glLightfv(GL_LIGHT0,  GL_SPECULAR, lightDifAndSpec);
   glLightfv(GL_LIGHT0,  GL_POSITION, lightPos);
   glEnable(GL_LIGHT0);  // Enable particular light source.

  // Material property vectors.
   float matSpec[] = { 1.0, 1.0, 1.0, 1.0 };
   float matDiffuse[] = {0.0, 1.0, 0.0, 1.0};
   float matShine[] = { 50.0 };

   // Material properties.
   glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffuse);
   glMaterialfv(GL_FRONT, GL_SPECULAR, matSpec);
   glMaterialfv(GL_FRONT, GL_SHININESS, matShine);


}

// Drawing routine.
void drawScene(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();

   gluLookAt(5.0, 10.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

   glPushMatrix();
   glRotatef(90,1,0,0);
   glutSolidTorus(2.0,  5.0, 80, 80);
   glTranslatef(15.0, 0.0, 0.0);
   glutSolidCube(6.0);
   glPopMatrix();
   glFlush();
}


// Main routine.
int main(int argc, char **argv)
{

   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);
   glutInitWindowSize(700, 700);
   glutInitWindowPosition(100, 100);
   glutCreateWindow("ballAndTorusReflected.cpp");
   glutDisplayFunc(drawScene);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glFrustum(-5.0, 5.0, -5.0, 5.0, 5.0, 100.0);
   setup();
   glutMainLoop();
}
```

- **2. Reflection 이용해 y축 대칭 시킨 glut obejcts 추가**

```
glPushMatrix();
  glScalef(1.0, -1.0, 1.0);
  glTranslatef(0, 10, 0);
  glRotatef(90,1,0,0);
  glutSolidTorus(2.0, 5.0, 80, 80);
  glTranslatef(15.0, 0.0, 0.0);
  glutSolidCube(6.0);
  glPopMatrix();
```

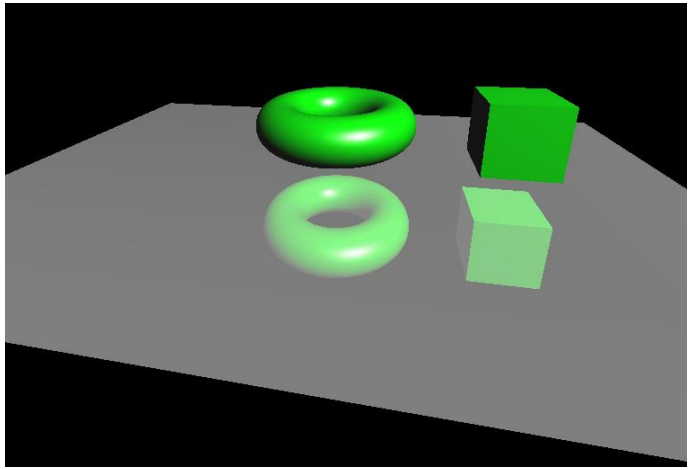


```
#include <iostream>

#include<GL/glut.h>

using namespace std;

void setup(void)
{
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glEnable(GL_DEPTH_TEST); // Enable depth testing.

   // Turn on OpenGL lighting.
   glEnable(GL_LIGHTING);

   // Light property vectors.
   float lightAmb[]  = { 0.0, 0.0, 0.0, 1.0 };
   float lightDifAndSpec[] = { 1.0, 1.0, 1.0, 1.0 };
   float lightPos[] = {30.0, 30.0, 30.0, 1.0 };

   // Light properties.
   glLightfv(GL_LIGHT0,  GL_AMBIENT, lightAmb);
   glLightfv(GL_LIGHT0,  GL_DIFFUSE, lightDifAndSpec);
   glLightfv(GL_LIGHT0,  GL_SPECULAR, lightDifAndSpec);
   glLightfv(GL_LIGHT0,  GL_POSITION, lightPos);
   glEnable(GL_LIGHT0);  // Enable particular light source.

  // Material property vectors.
   float matSpec[] = { 1.0, 1.0, 1.0, 1.0 };
   float matDiffuse[] = {0.0, 1.0, 0.0, 1.0};
   float matShine[] = { 50.0 };

   // Material properties.
   glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffuse);
   glMaterialfv(GL_FRONT, GL_SPECULAR, matSpec);
   glMaterialfv(GL_FRONT, GL_SHININESS, matShine);


}

// Drawing routine.
void drawScene(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();

   gluLookAt(5.0,  10.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

   glPushMatrix();
   glRotatef(90,1,0,0);
   glutSolidTorus(2.0,  5.0, 80, 80);
   glTranslatef(15.0, 0.0, 0.0);
   glutSolidCube(6.0);
   glPopMatrix();

  glPushMatrix();
   glScalef(1.0, -1.0, 1.0);
   glTranslatef(0, 10, 0);
   glRotatef(90,1,0,0);
   glutSolidTorus(2.0,  5.0, 80, 80);
   glTranslatef(15.0, 0.0, 0.0);
   glutSolidCube(6.0);
   glPopMatrix();

   glFlush();
}


// Main routine.
int main(int argc, char **argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA | GLUT_DEPTH);
   glutInitWindowSize(700, 700);
   glutInitWindowPosition(100, 100);
   glutCreateWindow("ballAndTorusReflected.cpp");
```

- **Lighting을 사용시에는 물체에 투명한 효과를 주기 위하여 물체 (material)의 (R,G,B,A) 중 A (alpha) 값을 조절하면 된다**

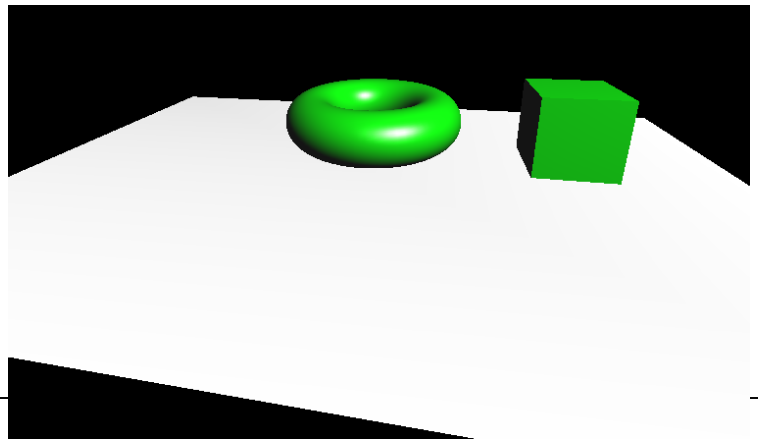
- 3. Lighting을 사용시 물체 (material)의 반사율 (R,G,B,A)을 사용하여 물체의 투명도를 설정하고 blending 효과를 줄 수 있다
- 물체의 전반적인 색에 가장 영향을 많이 미치는 diffuse 반사에 alpha값을 조절하였다. 반사되는 것처럼 보이기 위하여 중간에 크기가 변환된 glutSolidCube를 넣었고 diffuse 반사에 alpha값을 0.5로 주었다.

```
float matDiffuse1[] = {1.0, 1.0,1.0, 0.5};
glPushMatrix();
glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffuse1);
  glTranslatef(0.0, -3.2, 0.0);
  glScalef(2.0, 0.01,2.0);
  glutSolidCube(20.0);
  glPopMatrix();
```

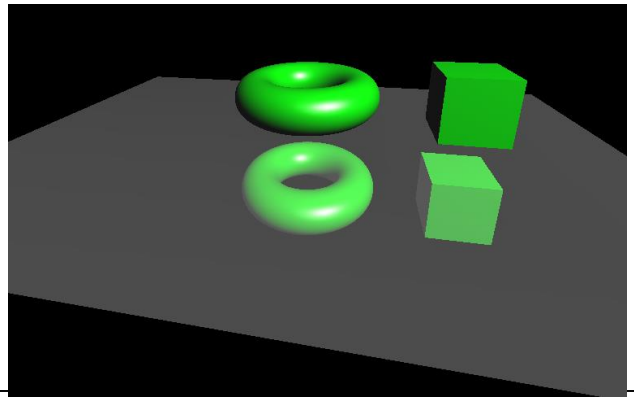


```
#include <iostream>

#include<GL/glut.h>

using namespace std;

void setup(void)
{
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glEnable(GL_DEPTH_TEST); // Enable depth testing.

   // Turn on OpenGL lighting.
   glEnable(GL_LIGHTING);

   // Light property vectors.
   float lightAmb[]  = { 0.2, 0.2, 0.2, 1.0 };
   float lightDifAndSpec[] = { 1.0, 1.0, 1.0, 1.0 };
   float lightPos[] = {30.0, 30.0, 30.0, 1.0 };

   // Light properties.
   glLightfv(GL_LIGHT0,  GL_AMBIENT, lightAmb);
   glLightfv(GL_LIGHT0,  GL_DIFFUSE, lightDifAndSpec);
   glLightfv(GL_LIGHT0,  GL_SPECULAR, lightDifAndSpec);
   glLightfv(GL_LIGHT0,  GL_POSITION, lightPos);
   glEnable(GL_LIGHT0);  // Enable particular light source.

  // Material property vectors.
   float matSpec[] = { 1.0, 1.0, 1.0, 1.0 };
   float matDiffuse[] = {0.0, 1.0, 0.0, 1.0};
   float matShine[] = { 50.0 };

   // Material properties.
   glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffuse);
   glMaterialfv(GL_FRONT, GL_SPECULAR, matSpec);
   glMaterialfv(GL_FRONT, GL_SHININESS, matShine);


}

// Drawing routine.
void drawScene(void)
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();

   gluLookAt(5.0, 10.0, 30.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);


   glPushMatrix();
   glRotatef(90,1,0,0);
   glutSolidTorus(2.0,  5.0, 80, 80);
   glTranslatef(15.0, 0.0, 0.0);
   glutSolidCube(6.0);
   glPopMatrix();

   glPushMatrix();
   glScalef(1.0, -1.0, 1.0);
   glTranslatef(0, 10, 0);
   glRotatef(90,1,0,0);
   glutSolidTorus(2.0,  5.0, 80, 80);
   glTranslatef(15.0, 0.0, 0.0);
   glutSolidCube(6.0);
   glPopMatrix();
   glEnable(GL_BLEND); // Enable blending.
   glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA); // Specify blending parameters.

   float matDiffuse1[] = {1.0, 1.0,1.0, 0.5};
   glPushMatrix();
   glMaterialfv(GL_FRONT, GL_DIFFUSE, matDiffuse1);
   glTranslatef(0.0, -3.2, 0.0);
   glScalef(2.0, 0.01,2.0);
   glutSolidCube(20.0);
   glPopMatrix();
   glFlush();
}
```

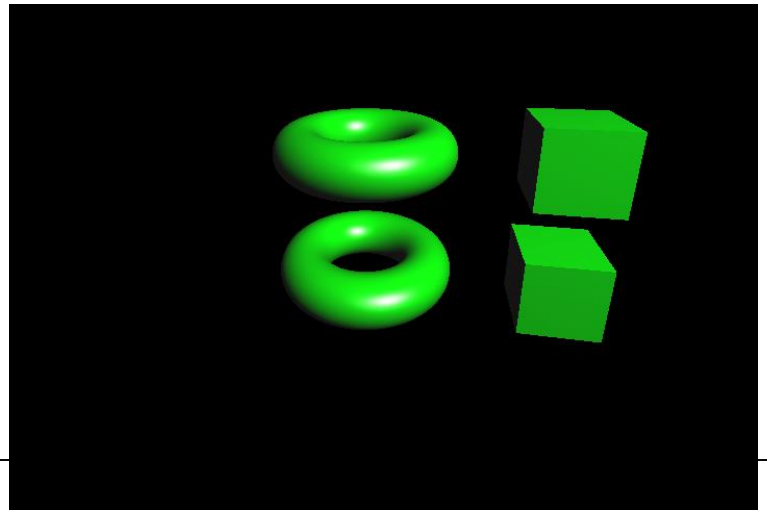
- 중간에 있는 물체 (cube)의 투명도에 따른 변화
- 1. float matDiffuse1[] = {1.0, 1.0,1.0, 1.0};
- Source의 alpha값 =1.0
- source 의 색:흰색, Destination (배경)의 색: 검정색
- $(R_f, G_f, B_f) = src_\alpha * (src_R, src_G, src_B) + (1 - src_\alpha) * (dst_R, dst_G, dst_B)$
- (R_f, G_f, B_f)=1.0*(1.0,1.0,1.0)+0.0*(0.0, 0.0, 0.0)
- (R_f, G_f, B_f)=(1.0,1.0,1.0) => 흰색



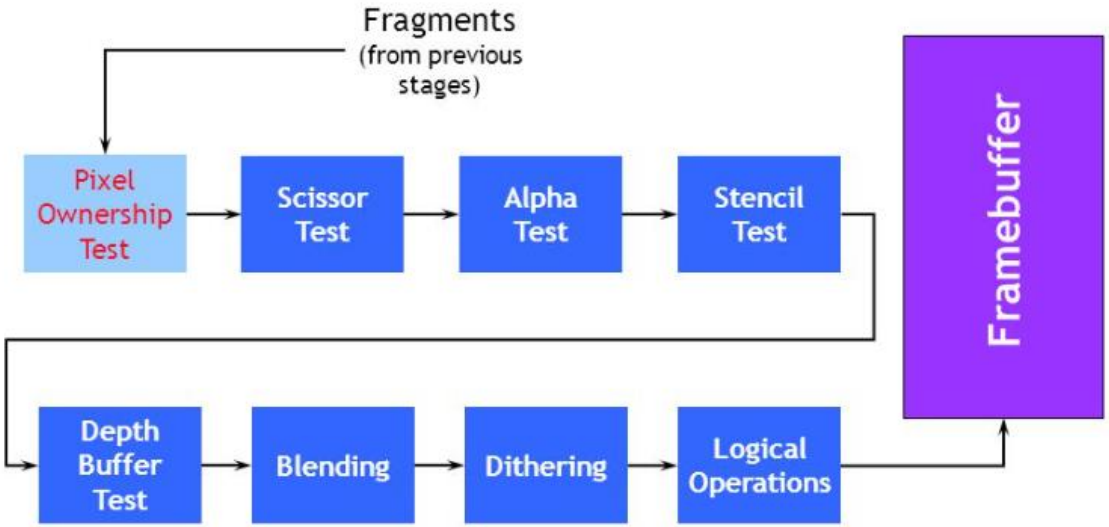
- 2. float matDiffuse1[] = {1.0, 1.0,1.0, 0.3};
- Source의 alpha값 =0.3
- source 의 색:흰색, Destination (배경)의 색: 검정색
- $(R_f, G_f, B_f) = src_\alpha * (src_R, src_G, src_B) + (1 - src_\alpha) * (dst_R, dst_G, dst_B)$
- (R_f, G_f, B_f)=0.3*(1.0,1.0,1.0)+0.7*(0.0, 0.0, 0.0)
- (R_f, G_f, B_f)=(0.3,0.3,0.3) => 회색



- 3. float matDiffuse1[] = {1.0, 1.0,1.0, 0.0};
- Source의 alpha값 =0.0
- source 의 색:흰색, Destination (배경)의 색: 검정색
- $(R_f, G_f, B_f) = src_\alpha * (src_R, src_G, src_B) + (1 - src_\alpha) * (dst_R, dst_G, dst_B)$
- (R_f, G_f, B_f)=0.0*(1.0,1.0,1.0)+1.0*(0.0, 0.0, 0.0)
- (R_f, G_f, B_f)=(0.0,0.0,0.0) => 검정색

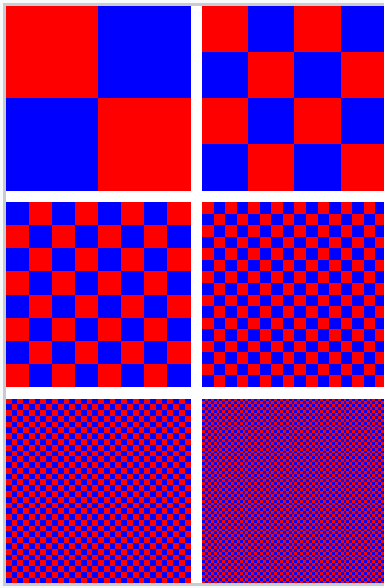


Fragment 연산: Dithering



- **Dithering**
- **Frame buffer의 용량이 적다면 dithering을 통해 다양한 색을 표현할 수 있다. 예를 들어 분홍색을 표현하는 경우 픽셀에 분홍색을 직접 주는 것이 아니라 빨간색과 흰색을 번갈아 가면서 바둑판 모양으로 칠해서 결국 눈에 보이는 것은 두 색의 평균에 해당하는 분홍색이 보이도록 하는 것. 실제 픽셀 값은 빨간색과 흰색 뿐임**
- **Dithering is a process whereby a limited colour palette is used to trick the eye into thinking there are more colours (or detail) within am image. A good example are old 16 colour computer games, where different combinations where used to make it look like there where more colours. A lot of printers use dithering to make an image look better**
- **OpenGL에서는 default로 켜져 있다**

```
glEnable(GL_DITHER);  // dithering 킨다
glDisable(GL_DITHER); // dithering 끈다
```



An illustration of dithering. Red and blue are the only colors used but, as the red and blue squares are made smaller, the patch appears purple.

- **Logical operations**
- **Fragment에 가해지는 최종 작업으로 blending과 유사하게 frame buffer에 있는 fragment (destination, d)의 색과 새로운 fragment (source, s)의 색과의 논리 연산을 나타낸다**
- **A logical operation can be applied between the fragment and the value stored at the corresponding location in the framebuffer; the result replaces the current framebuffer value. You choose the desired logical operation with glLogicOp. Logical operations are performed only on color indexes**