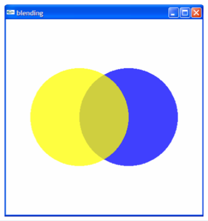
Computer Graphics

Transparency Case Study

taken from OpenGL Lectures by Tom Duff and George Ledin Jr.

How is transparency achieved in OpenGL?

- In OpenGL, we use blending of alpha value (opacity value) to create a translucent fragment that lets some of the previously stored color value "show through".



Enable/Disable blending of color

- To enable transparency, we need to explicitly enable blending:
  - glEnable(GL_BLEND)
- To disable blending:
  - glDisable(GL_BLEND)

Alpha Value

- void **glClearColor**(GLclampf *red*, GLclampf *green*,
- GLclampf *blue*, GLclampf *alpha*)
  - Specify values for the color buffer
- void **glColor4f**(GLfloat *red*, GLfloat *green*, GLfloat *blue*, GLfloat *alpha*)
  - Specify colors

0 <= alpha <= 1

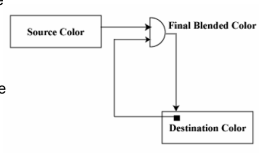
Transparent opaque

What happens when blending is enabled?

- When blending is enabled, the alpha value is used to combine the color value of the fragment being processed with that of the pixel already stored in the frame buffer.
  - Blending occurs after the scene has been rasterized and converted to fragments, but just before the final pixels are drawn in the frame buffer.

- **Without blending**, each new fragment overwrites any existing color values in the frame buffer, as if the fragment were opaque.
- **With blending**, one can control how much of the existing color value should be combined with the new fragment's value.

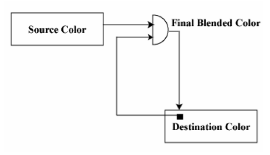
Blending Model



The source and destination factors

- During blending, source is the color value of the incoming fragment.
- Destination is the currently stored pixel value.

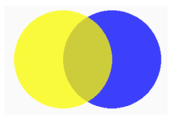
Blending Model



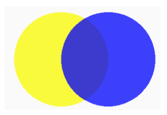
Case Study 1

- Scenario:
  - There's a yellow circle and a blue circle. The yellow circle is 25% transparent. The blue circle is 75% transparent.
- Question:
  - When the yellow circle is in front of the blue circle, what do we see, Choice A or Choice B?

- Choice A



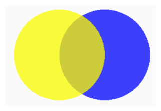
- Choice B



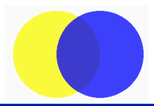
Case Study 1 – Answer is A

- **Reason:** Because yellow circle is more opaque, therefore we see more of the yellow and less of the blue.
  - Scenario:
    - There's a yellow circle and blue circle.
      - The yellow circle is 25% transparent. The blue circle is 75% transparent.
      - This means that the yellow circle is 75% opaque and blue circle is 25% opaque.
  - If the yellow circle is in front of the blue circle, only 25% of the blue blends with 75% of the yellow. (Choice A)
  - If the blue circle is in front of the yellow circle, only 25% of yellow blends with 75% of the blue. (Choice B)

- Choice A



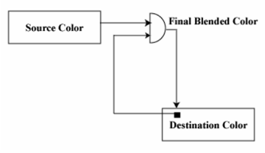
- Choice B



The source and destination factors

- During blending, source is the color value of the incoming fragment.
- Destination is the currently stored pixel value.

Blending Model



How does OpenGL blend the source and destination color values?

- Assume: Source Color: (Rs, Gs, Bs, As)
  Destination Color: (Rd, Gd, Bd, Ad)
- Step 1:
  - Specify the source and destination factors.
    - Let source destination blending factors be: – (Sr, Sg, Sb, Sa)
    - Let destination blending factors be: – (Dr, Dg, Db, Da)
- Step 2:
  - Combine the corresponding components of source and destination.
  - Final blended RGBA values are given by:
    - (RsSr+RdDr, GsSg+GdDg, BsSb+BdDb, AsSa+AdDa)
- Note: Each of these quadruplets is clamped to [0,1]

Blend Function

void **glBlendFunc**(GLenum *sfactor*, GLenum *dfactor*)

- *dfactor:* Specifies how the red, green, blue and alpha destination blending factors are computed. 8 symbolic constants are accepted:
  - GL_ZERO
  - GL_ONE
  - GL_SCR_COLOR
  - GL_ONE_MINUS_SRC_COLOR
  - GL_SRC_ALPHA
  - GL_ONE_MINUS_SRC_ALPHA
  - GL_DST_ALPHA
  - GL_ONE_MINUS_DST_ALPHA.
- *sfactor*: Specifies how the red, green, blue and alpha source blending factors are computed. 9 symbolic constants are accepted:
  - GL_ZERO
  - GL_ONE
  - GL_DST_COLOR
  - GL_ONE_MINUS_DST_COLOR
  - GL_SRC_ALPHA
  - GL_ONE_MINUS_SRC_COLOR
  - GL_DST_ALPHA
  - GL_ONE_MINUS_DST_ALPHA
  - GL_SRC_ALPHA_SATURATE

Source and Destination Blending Factors Table

sFactor=(Sr,Sg,Sb,Sa) or dFactor=(Dr,Dg,Db,Da)

Constant	Relevant Factor	Computed Blend Factor
GL_ZERO	source or destination	(0, 0, 0, 0)
GL_ONE	source or destination	(1, 1, 1, 1)
GL_DST_COLOR	source	(R_d, G_d, B_d, A_d)
GL_SRC_COLOR	destination	(R_s, G_s, B_s, A_s)
GL_ONE_MINUS_DST_COLOR	source	$(1, 1, 1, 1)-(R_d, G_d, B_d, A_d)$
GL_ONE_MINUS_SRC_COLOR	destination	$(1, 1, 1, 1)-(R_s, G_s, B_s, A_s)$
GL_SRC_ALPHA	source or destination	(A_s, A_s, A_s, A_s)
GL_ONE_MINUS_SRC_ALPHA	source or destination	$(1, 1, 1, 1)-(A_s, A_s, A_s, A_s)$
GL_DST_ALPHA	source or destination	(A_d, A_d, A_d, A_d)
GL_ONE_MINUS_DST_ALPHA	source or destination	$(1, 1, 1, 1)-(A_d, A_d, A_d, A_d)$
GL_SRC_ALPHA_SATURATE	source	(f, f, f, 1); f=min(As, 1-Ad)

Case Study 2: Source and Destination Blending Factors

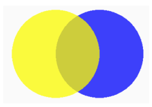
- **Case Study Setup:**
  - **A yellow circle of radius 2, centered at (-1,0,0)**

```
void drawLeftCircle()  {
 /* draw yellow circle on LHS of screen */
 glColor4f(1.0, 1.0, 0.0, 0.75); //yellow, alpha=0.75
 circle(-1,0,2); //x=-1,y=0,radius=2
}
```

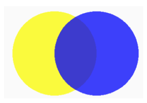
  - **A blue circle of radius 2, centered at (1,0,0)**

```
void drawRightCircle(){
 /* draw yellow triangle on LHS of screen */
 glColor4f(0.0, 0.0, 1.0, 0.75); //blue, alpha=0.75
 circle(1,0,2); //x=1,y=0,radius=2
}
```

- Choice A



- Choice B



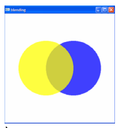
- **Question:**
  - If we change the sFactor and dFactor of glBlendFunc(sFactor, dFactor), what will be the blending effect?

Init blending function in main

```
/* We will change sFactor and dFactor values later on to test the blending effect.*/

//(Sr,Sg,Sb,Sa)=(As,As,As,As)= (0.75,0.75,0.75,0.75)
sFactor = GL_SRC_ALPHA;

//(Dr,Dg,Db,Da)=(1-As,1-As,1-As,1-As)= (0.25,0.25,0.25,0.25)
dFactor = GL_ONE_MINUS_SRC_ALPHA;

/* Initialize alpha blending function. */
void init()
{
     glEnable (GL_BLEND);
     glBlendFunc (sFactor, dFactor);
     glShadeModel (GL_FLAT);
     glClearColor (1, 1, 1, 1); // white and opaque background
}
```

Test Case A : Let sFactor = GL_SRC_ALPHA, dFactor = GL_ONE_MINUS_SRC_ALPHA Draw blue circle first, yellow second.

```
void display() {
     drawRightCircle(); // blue circle
     drawLeftCircle(); // yellow circle
}
```



- Predicted final blending value based on formula:
  (Rf,Gf,Bf,Af) = (RsSr+RdDr, GsSg+GdDg, BsSb+BdDb, AsSa+AdDa)

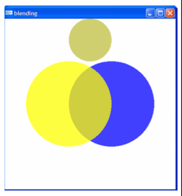
  - (Rs,Gs,Bs,As) = (1,1,0,0.75)   //yellow, semi-transparent
  - (Rd,Gd,Bd,Ad) = (0,0,1,0.75) //blue, semi-transparent
  - sFactor = GL_SRC_ALPHA
    - (Sr,Sg,Sb,Sa) = (0.75,0.75,0.75,0.75)
  - dFactor = GL_ONE_MINUS_SRC_ALPHA
    - (Dr,Dg,Db,Da) = (1-0.75,1-0.75,1-0.75,1-0.75) = (0.25,0.25,0.25,0.25)

- Final predicted blending value (Rf,Gf,Bf,Af) = (0.75,0.75,0.25,0.75)

Use the predicted final blending value to draw an object (circle) and compare its color with the blended color of the blue and yellow circles

- In the preceding slide, our predicted final blending value was (Rf,Gf,Bf,Af) = (0.75,0.75,0.25,0.75)
- Function to draw a circle centered at (0,3,0) with radius of 1. Display the test circle together with two intersecting circles

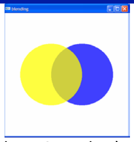
```
void drawTestCircle() {
     glColor4f(0.75, 0.75, 0.25, 0.75);
     circle(0,3,1);   }
void display() {
     drawTestCircle(); // draw test circle with predicted color
     drawRightCircle(); // Destination: blue circle
     drawLeftCircle(); // Source: yellow circle
}
```



- Result:
  - The predicted color shown in the test circle matches the final blended color of the yellow and blue circles.

Test Case B : Let sFactor = GL_SRC_ALPHA, dFactor = GL_ONE_MINUS_SRC_ALPHA Draw yellow circle first, blue circle second

```
void display() {
      drawLeftCircle(); // yellow circle
      drawRightCircle(); // blue circle
}
```

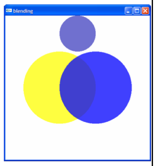


- Predicted final blending value based on formula:
  (Rf,Gf,Bf,Af) = (RsSr+RdDr, GsSg+GdDg, BsSb+BdDb, AsSa+AdDa)
  - (Rs,Gs,Bs,As) = (0,0,1,0.75) //blue, semi-transparent
  - (Rd,Gd,Bd,Ad) = (1,1,0,0.75) //yellow, semi-transparent
  - sFactor = GL_SRC_ALPHA
    - (Sr,Sg,Sb,Sa) = (0.75,0.75,0.75,0.75)
  - dFactor = GL_ONE_MINUS_SRC_ALPHA
    - (Dr,Dg,Db,Da) = (1-0.75,1-0.75,1-0.75,1-0.75) = (0.25,0.25,0.25,0.25)
- Final predicted blending value (Rf,Gf,Bf,Af) = (0.25,0.25,0.75,0.75)

Use the predicted final blending value to draw an object (circle) and compare the color with the blending color of the yellow and blue circles.

- In the preceding slide, our predicted final blending value was
  (Rf,Gf,Bf,Af) = (0.25,0.25,0.75,0.75)
- Function to draw a circle centered at (0,3,0) with radius of 1.

```
void drawTestCircle()
{ glColor4f(0.25, 0.25, 0.75, 0.75);
  circle(0,3,1);}
```



- Display the test circle together with two intersecting circles

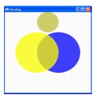
```
void display()
{ drawTestCircle();
  drawLeftCircle();
  drawRightCircle(); }
```

- Result:
  - The predicted color shown in the test circle matches the final blending color of the yellow and blue circle.

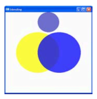
Conclusion drawn from Test A and Test B

- In both TestA and B, we use the same blending function:
  - glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
- The differences are:
  - In Test A, we draw first the blue, then the yellow circle.
  - In Test B, we draw first the yellow, then blue circle.
- Tests A and B display different blending effects.
- This shows that the order in which objects are defined makes a difference in the final color blending result.

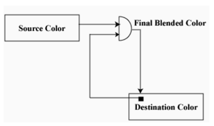
Test a



Test b



Blending Model



Case 3 Blending is dangerous and has many artifacts

Study Case:

- A cube emitting blue light.

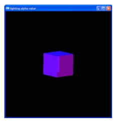
```
GLfloat mat_emission[] = { 0, 0, 1, 1 }; // blue
glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);//blue
```

- When light shines on the cube, the material reflects red diffuse color of alpha value 0.6.

```
GLfloat mat_transparent1[] = { 1, 0, 0, 0.8}; // red 0.8 diffuse
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_transparent1);//red light
```

- One light shining in the –Z direction

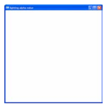
```
void init() {
      GLfloat position[] = { 0, 0, 1.0, 0 }; //light shining headon
      glLightfv(GL_LIGHT0, GL_POSITION, position);
      glEnable(GL_LIGHTING);
      glEnable(GL_LIGHT0);
      glEnable(GL_DEPTH_TEST);
}
```



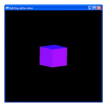
Case 3 A
Object is visible with black background but disappears with white background

```
void display() {
        GLfloat mat_zero[] = { 0.0, 0.0, 0.0, 1.0 }; //black
        GLfloat mat_transparent[] = { 1, 0, 0, 0.6}; //red 0.6 diffuse
        GLfloat mat_emission[] = { 0.0, 0, 1, 1 }; // blue

        SetBackground(1,1,1,1);
        glRotatef (15.0, 1.0, 1.0, 0.0);
        glRotatef (30.0, 0.0, 1.0, 0.0);
        glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);//blue
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_transparent);//red 0.6 diffuse
        glEnable (GL_BLEND);
        glDepthMask (GL_FALSE);// make the depth buffer read only
                        // while drawing the translucent objects
        glBlendFunc (GL_SRC_ALPHA, GL_ONE);
        glCallList (cubeList);
        glDepthMask (GL_TRUE);
        glDisable (GL_BLEND);
        glFlush();
}
```

White background
SetBackground(1,1,1,1);



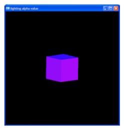
Black background
SetBackground(1,1,1,1);



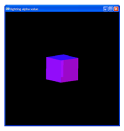
Case 3B Blending with depth buffer read-only vs depth buffer writable

```
void display() {
        GLfloat mat_zero[] = { 0.0, 0.0, 0.0, 1.0 }; //black
        GLfloat mat_transparent[] = { 1, 0, 0, 0.6}; //red 0.6 diffuse
        GLfloat mat_emission[] = { 0.0, 0, 1, 1 }; // blue

        SetBackground(0,0,0,1);
        glRotatef (15.0, 1.0, 1.0, 0.0);
        glRotatef (30.0, 0.0, 1.0, 0.0);
        glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_transparent);
        glEnable (GL_BLEND);
        glDepthMask (GL_FALSE);// make the depth buffer read only
                        // while drawing the translucent objects
        glBlendFunc (GL_SRC_ALPHA, GL_ONE);
        glCallList (cubeList);
        glDepthMask (GL_TRUE);
        glDisable (GL_BLEND);
        glFlush();
}
```

With depth buffer read-only
glDepthMask (GL_FALSE);



With depth buffer writable
glDepthMask (GL_TRUE);

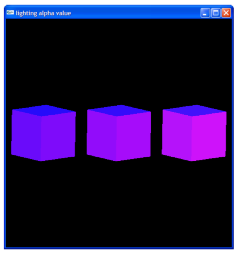


Lesson from Case 3

- Blending is dangerous mixed with light whose alpha value is < 1 (transparent).
- In order to avoid artifacts, you need to know exactly what you are doing.

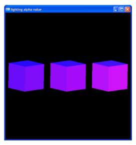
Case 4 How does the alpha value affect the object's color?

- Study Case:
  - There are three cubes emitting blue color.
  - If light is shine on the left, it will present red diffuse color of alpha value, such as 0.6 transparency.
  - If light is shined on the middle, it will present red diffuse color of alpha value, such as 0.8 (transparency).
  - If light is shine on the right, it will present red diffuse color of alpha = 1 (solid).
- Question:
  - How does the alpha value of light affect the color of the cube?



Case Study 4: Setup common traits for the three cubes to be rendered

```
void MyInit()
{
        GLfloat mat_specular[] = { 0.1, 0.1, 0.1, 1 } ; //dark gray specular light
        GLfloat mat_shininess[] = { 127 }; // concentrated specular spot
        GLfloat position[] = { 0, 0, 1.0, 0 };   //  toward –Z directional light

        glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
        glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);        //light shining head on
        glLightfv(GL_LIGHT0, GL_POSITION, position);
        glEnable(GL_LIGHTING);
        glEnable(GL_LIGHT0);         //one light source
        glEnable(GL_DEPTH_TEST);

        // create display list for a cube to be used later
        cubeList = glGenLists(1);
        glNewList(cubeList, GL_COMPILE);
        glutSolidCube (0.6);
        glEndList();
}
```



Case Study 4, continued

```
void display()
{
        GLfloat mat_transparent1[] = { 1, 0, 0, 0.6}; // red 0.6 diffuse
        GLfloat mat_transparent2[] = { 1, 0, 0, 0.8}; // red 0.8 diffuse
        GLfloat mat_transparent3[] = { 1, 0, 0, 1}; // red 1 diffuse
        GLfloat mat_emission[] = { 0.0, 0, 1, 1 }; // blue
        SetBackground(0,0,0,1); //black

        // Cube on the left: diffuse = red, alpha = 0.6
        glPushMatrix ();
        glTranslatef (-1, 0, 0);     glRotatef (15.0, 1.0, 1.0, 0.0);    glRotatef (30.0, 0.0, 1.0, 0.0);
        glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);//blue
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_transparent1);//red
        glEnable (GL_BLEND);
        glBlendFunc (GL_SRC_ALPHA, GL_ONE);
        glCallList (cubeList);    // use  a display list to draw the cubes
        glDisable (GL_BLEND);
        glPopMatrix ();
        glFlush();
```

Case Study 4, continued

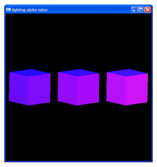
```
// Cube in the middle: diffuse = red, alpha = 0.8
glPushMatrix ();
glTranslatef (0, 0, 0);
glRotatef (15.0, 1.0, 1.0, 0.0);
glRotatef (30.0, 0.0, 1.0, 0.0);
glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);//blue
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_transparent2);//red
glEnable (GL_BLEND);
glBlendFunc (GL_SRC_ALPHA, GL_ONE);
glCallList (cubeList);
glDepthMask (GL_TRUE);
glDisable (GL_BLEND);
glPopMatrix ();
```

Case Study 4, continued

```
        //Cube on the right: diffuse=red, alpha = 1
        glPushMatrix ();
        glTranslatef (1, 0, 0);
        glRotatef (15.0, 1.0, 1.0, 0.0);
        glRotatef (30.0, 0.0, 1.0, 0.0);
        glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);  //blue
        glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_transparent3);  //red

        glEnable (GL_BLEND);
        glBlendFunc (GL_SRC_ALPHA, GL_ONE);
        glCallList (cubeList);
        glDisable (GL_BLEND);
        glPopMatrix ();

        glFlush();
}
```

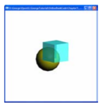
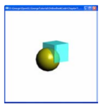
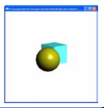
Case Study 4 Conclusions

- The alpha value of material affects the final object color.
- Between [0,1], the larger the alpha value, the Deeper the color.
- In this case, the cube on the right has alpha value 1 for its diffuse color, therefore it diffuses more red. Since red (diffuse) + blue (emissive color) = purple, therefore, the cube on the right displays deeper purple.



Case Study 5: Blending with depth buffer

- Scenario:
  - In the scene, there is a translucent cyan cube in front of a solid yellowish sphere.
    - The cube is centered at (0.1, 0.1, 8)
    - The sphere is centered at (-0.1, -0.1, 8)
  - Each time user click "a" on the keyboard, the cube will move backwards a bit.
  - When user click "r" on the keyboard, the cube will be reset to its original position.
- Goal:
  - **The part of translucent cyan cube that is in front of the solid sphere blended with sphere.**
  - **The part of translucent cyan cube that is behind the sphere will not be displayed.**

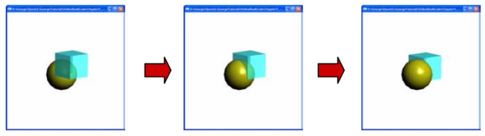
Case study 5 cont.

```
#define ZINC 1
#define MAXZ 8.0
#define MINZ -8.0
static float solidZ = MINZ;
static float transparentZ = MAXZ;

void ChangeLocation()
{
      transparentZ -= ZINC;
      glutPostRedisplay();
}
```

```
void keyboard(unsigned char key,
                          int x, int y) {
   switch (key) {
   case 'a':
   case 'A': ChangeLocation();
              glutPostRedisplay();
              break;
   case 'r':
   case 'R': transparentZ = MAXZ;
              glutPostRedisplay();
              break;
   case 27: exit(0);
   }
}
```

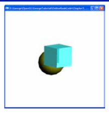
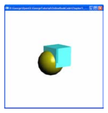
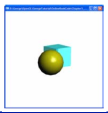
Case study 5 cont.

- When user click 'a' or 'A", the global valuable which keeps track of the location of the cyan cube is decremented.
- The result is that the cube is moved backward.
- As the transparent cube moves backward, **what we expect to see** is that it blends with the opaque sphere.



Problem: when the translucent cyan cube is in the front of solid sphere, the color of cube does not blend with the color of the sphere

```
void display() { …
        //glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        SetBackground(1,1,1,1);
        // Draw the translucent cube farther from camera
        glPushMatrix ();
        glTranslatef (0.15, 0.15, transparentZ);
        …
        glEnable (GL_BLEND);
        glBlendFunc (GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
        glCallList (cubeList);
        glDisable (GL_BLEND);
        glPopMatrix ();
        // Draw the solid sphere closer to camera glPushMatrix ();
        glTranslatef (-0.15, -0.15, solidZ);
         …
        glCallList (sphereList); // draw the solid sphere
        glPopMatrix ();
        glutSwapBuffers();
}
```

How to fix the problem?

- If you want translucent objects (in front) blending with solid objects, and solid objects (in front) obscuring translucent objects, you need to exercise care if you draw the translucent and solid objects in one scene.
- The way to draw the objects are as follows:

(1) Enable the depth buffer

(2) Draw all opaque objects

(3) Enable blend, then draw all translucent objects.

The fix

```
void display()  {
         …
        //glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        SetBackground(1,1,1,1);
        // Draw the solid sphere closer to camera
        glPushMatrix ();
        glTranslatef (-0.15, -0.15, solidZ);        …
        glCallList (sphereList); // draw the solid sphere
        glPopMatrix ();

        // Draw the translucent cube farther from camera glPushMatrix ();
        glTranslatef (0.15, 0.15, transparentZ);        …
        glEnable (GL_BLEND);
        glBlendFunc (GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
        glCallList (cubeList);
        glDisable (GL_BLEND);
        glPopMatrix ();
        glutSwapBuffers();
}
```