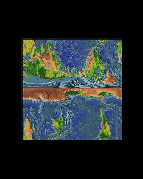
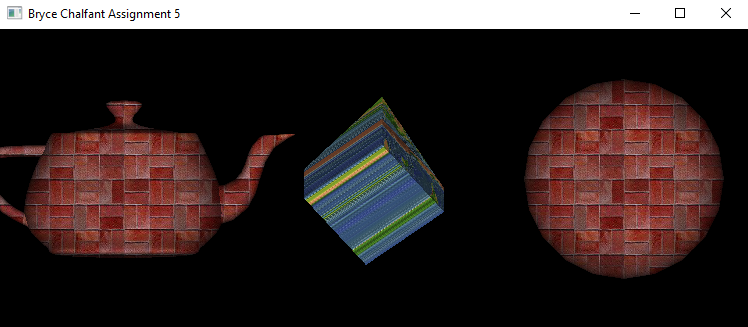
Cube:

With my code, I create a solid cube with the size of one and have it spawn in the center of the window. I also bind the texture Earth-1024X512.jpg to the cube after loading it in through freeimage library and linearly map it. I did this by specifying that the texturing should be GL\_OBJECT\_LINEAR on both the T and S variables. This causes the mapping of the texture to be one to one with the object face. This causes no warping compared to if you did a texture mapping of a sphere.



Arcball:

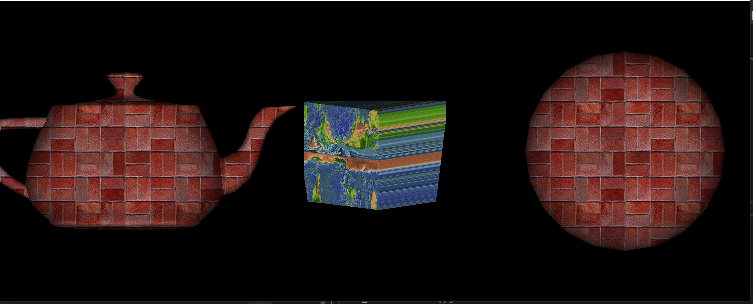
I did archball by having several different helper functions such as recomputeOrientation which recomputes the x, y, and z values that I used in my glutLookAt. When you left click on the screen it send the location of mouse to the mouseCallback and saves the state and then sets the location of the mouse to mousex and mousey to the mouseMotion function where it computes the phi and theta of the camera then it recomputes where the camera should look at. I also placed the glutLookAt right before I spawned the cube so that way the cube and the camera are in the same matrix. This will cause the cube to spin around in place while the teapot and sphere stay idle.



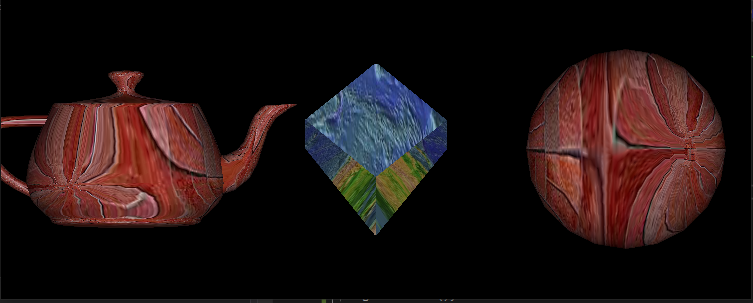
Sphere VS Linear:

So with my code, I spawned in a sphere with glutSolidSphere with a radius of 1, and a stack and slice of 20. I also spawned a teapot with glutSolidTeapot and applied the same brick texture to them. I assigned the texturing to be GL\_SPHERE\_MAP for both the T and S variables which means that the textures instead of being linear mapping it’s based off how it would be mapped on a sphere. This causes some warping depending on how the mapping is. When I switched the texture mapping to GL\_OBJECT\_LINEAR the textures did not warp any more, but if you were to rotate the cube with the linear the sides would have the same color as the last pixel on the edges and the color will stay the same until it reaches the other edge. The difference that I noticed was that with sphere mapping you can cover the whole object with the texture, but it causes a weird warp of the texture and with linear mapping it doesn’t create a warping affect, but it only maps it to the front and back of the object. All other sides will have the same colors as the edge pixels and it will continue until the other edge.

Linear object



Sphere



// this program demonstrates an interactive program that uses glColorMaterial()

//to change material parameters. Pressing each of the three mouse buttons changes the color of the diffuse reflection.

// the cube demonstrates a rotating light whenever the mouse left button clicked

#include <iostream>

#include <GL/glut.h>

#include <GL/freeglut.h>

#include <FI/FreeImage.h>

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#define PI 3.141592

GLint leftMouseButton, rightMouseButton; //status of the mouse buttons

int mouseX = 0, mouseY = 0; //last known X and Y of the mouse

float cameraTheta, cameraPhi, cameraRadius; //camera position in spherical coordinates

float x, y, z;

GLuint texID[2]; // Texture ID's for the three textures.

char\* textureFileNames[2] = { // file names for the files from which texture images areloaded

(char\*)"textures/The Sun.jpg",

(char\*)"textures/The Earth.jpg",

};

static bool active = false;

static int spin = 0;

int speed = 1000 / 60;

void recomputeOrientation()

{

x = cameraRadius \* sinf(cameraTheta)\*sinf(cameraPhi);

z = cameraRadius \* cosf(cameraTheta)\*sinf(cameraPhi);

y = cameraRadius \* cosf(cameraPhi);

glutPostRedisplay();

}

void init(void)

{

glClearColor(0.0, 0.0, 0.0, 0.0);

glShadeModel(GL\_SMOOTH);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_DEPTH\_TEST);

}

/\* Here is where the light position is reset after the modeling

\* transformation (glRotated) is called. This places the

\* light at a new position in world coordinates. The cube

\* represents the position of the light.

\*/

void loadTextures() {

int i;

glGenTextures(2, texID); // Get the texture object IDs.

for (i = 0; i < 2; i++) {

void\* imgData; // Pointer to image color data read from the file.

int imgWidth; // The width of the image that was read.

int imgHeight; // The height.

FREE\_IMAGE\_FORMAT format =

FreeImage\_GetFIFFromFilename(textureFileNames[i]);

if (format == FIF\_UNKNOWN) {

printf("Unknown file type for texture image file %s\n",

textureFileNames[i]);

continue;

}

FIBITMAP\* bitmap = FreeImage\_Load(format, textureFileNames[i], 0); // Readimage from file.

if (!bitmap) {

printf("Failed to load image %s\n", textureFileNames[i]);

continue;

}

FIBITMAP\* bitmap2 = FreeImage\_ConvertTo24Bits(bitmap); // Convert to RGBor BGR format

FreeImage\_Unload(bitmap);

imgData = FreeImage\_GetBits(bitmap2); // Grab the data we need from thebitmap.

imgWidth = FreeImage\_GetWidth(bitmap2);

imgHeight = FreeImage\_GetHeight(bitmap2);

if (imgData) {

printf("Texture image loaded from file %s, size %dx%d\n",

textureFileNames[i], imgWidth, imgHeight);

glBindTexture(GL\_TEXTURE\_2D, texID[i]); // Will load image dat into texture object #i

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, imgWidth, imgHeight, 0,

GL\_RGB,

GL\_UNSIGNED\_BYTE, imgData);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR); //Required since there are no mipmaps.

}

else {

printf("Failed to get texture data from %s\n", textureFileNames[i]);

}

}

}

void mouseWheel(int wheel, int direction, int x, int y) {

if (direction > 0) {

z -= (z \* .2);

std::cout << z;

glutPostRedisplay();

}

else {

z += (z \* .2);

glutPostRedisplay();

}

//attempting to move the camera to and away from the model on the z axis

}

void display(void)

{

GLfloat position[] = { 0.0, 0.0, 1.5, 2.0};

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glTexGeni(GL\_S, GL\_TEXTURE\_GEN\_MODE, GL\_SPHERE\_MAP);

glTexGeni(GL\_T, GL\_TEXTURE\_GEN\_MODE, GL\_SPHERE\_MAP);

glEnable(GL\_TEXTURE\_2D);

glEnable(GL\_TEXTURE\_GEN\_S);

glEnable(GL\_TEXTURE\_GEN\_T);

glShadeModel(GL\_SMOOTH);

glBindTexture(GL\_TEXTURE\_2D, texID[0]);

glPushMatrix();

glTranslatef(0.0, 0.0, -5.0);

gluLookAt(x, y, z, 0, 0, 0, 0, 1, 0);

glPushMatrix();

glRotated((GLdouble)spin, 0.0, 1.0, 0.0);

glLightfv(GL\_LIGHT0, GL\_POSITION, position);

glTranslated(0.0, 0.0, 1.5);

glDisable(GL\_LIGHTING);

glutSolidSphere(.1, 20, 20);

glEnable(GL\_LIGHTING);

glPopMatrix();

glBindTexture(GL\_TEXTURE\_2D, texID[1]);

glutSolidSphere(.5, 20, 20);

glPopMatrix();

glutSwapBuffers();

}

void reshape(int w, int h)

{

glViewport(0, 0, (GLsizei)w, (GLsizei)h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(40.0, (GLfloat)w / (GLfloat)h, 1.0, 20.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

void mouse(int button, int state, int x, int y)

{

switch (button) {

case GLUT\_RIGHT\_BUTTON:

if (state == GLUT\_DOWN) {

spin = (spin + 30) % 360;

glutPostRedisplay();

}

break;

case GLUT\_LEFT\_BUTTON:

leftMouseButton = state;

mouseX = x;

mouseY = y;

default:

break;

}

}

void mouseMotion(int x, int y)

{

if (leftMouseButton == GLUT\_DOWN)

{

cameraTheta += (mouseX - x)\*0.005;

cameraPhi += (mouseY - y)\*0.005;

// make sure that phi stays within the range (0, PI)

if (cameraPhi <= 0)

cameraPhi = 0 + 0.001;

if (cameraPhi >= PI)

cameraPhi = PI - 0.001;

recomputeOrientation(); //update camera (x,y,z)

}

// camera zoom in/out

mouseX = x;

mouseY = y;

}

void keyboard(unsigned char key, int x, int y) {

if (key == 'a' || key == 'A') {

active = !active;

}

}

void specialInputs(int key, int x, int y) {

switch (key)

{

case GLUT\_KEY\_UP:

speed -= 1;

if (speed == 0) speed = 1;

break;

case GLUT\_KEY\_DOWN:

speed += 1;

break;

default:

break;

}

}

void timer(int value) {

if(active)

{

spin++;

glutPostRedisplay();

}

glutTimerFunc(speed, timer, 0);

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_SINGLE | GLUT\_RGB | GLUT\_DEPTH);

glutInitWindowSize(500, 500);

glutInitWindowPosition(100, 100);

glutCreateWindow(argv[0]);

init();

loadTextures();

cameraRadius = 3.0f;

cameraTheta = 1;

cameraPhi = 1;

recomputeOrientation();

glutDisplayFunc(display);

glutReshapeFunc(reshape);

glutMouseFunc(mouse);

glutMotionFunc(mouseMotion);

glutKeyboardFunc(keyboard);

glutSpecialFunc(specialInputs);

glutMouseWheelFunc(mouseWheel);

glutTimerFunc(1000/60, timer, 0);

glutMainLoop();

return 0;

}

// This example demonstrates how to do texture mapping in opengl with the freeimage lib

// you can download the texture image files used in this program from http://borg.csueastbay.edu/~bhecker/Previous%20Terms/CS-455-Fall18 / Textbook / source / glut / textures /

#include <GL/glut.h>

#include <stdio.h>

#include <iostream>

#include <FI/freeimage.h>

/\* Draws a simple scene to show how to use texture objects. Three texture images

\* are loaded into three texture objects. The scene contains three copies of the

\* GLUT teapot, using the three different textures.

\* The program uses the FreeImage image processing library to read the texture

\* images from files.

\*/

// --------------------------------- global variables --------------------------------

GLuint texID[3]; // Texture ID's for the three textures.

char\* textureFileNames[3] = { // file names for the files from which texture images are loaded

(char\*)"textures/Earth-1024x512.jpg",

(char\*)"textures/brick001.jpg",

(char\*)"textures/marble.jpg"

};

#define PI 3.141592

GLint leftMouseButton, rightMouseButton; //status of the mouse buttons

int mouseX = 0, mouseY = 0; //last known X and Y of the mouse

float cameraTheta, cameraPhi, cameraRadius; //camera position in spherical coordinates

float x, y, z;

// Note that you need to make a folder named “textures” and then copy the image files to

// the folder.

float rotateX = 0; // rotations for a simple viewing transform, applied to

float rotateY = 0; // each teapot, controlled by the arrow and HOME keys.

// ------------------------ OpenGL initialization and rendering -----------------------

void initGL() {

// called by main() to initialize the OpenGL drawing context

glClearColor(0.0, 0.0, 0.0, 1.0); // background color

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

float white[4] = { 1, 1, 1, 1 }; // A white material, suitable for texturing.

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE, white);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-3.75, 3.75, -1.5, 1.5, -2, 2);

glMatrixMode(GL\_TEXTURE); // Matrix mode for manipulating the texture transform matrix.

glLoadIdentity();

glScalef(1, -1, 1); // It turns out the texture coordinates for the GLUT teapotput an upside

// down texture on the sides of the teapot. To fix that, I apply a texture

// transform that flips the texture vertically.

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

} // end initGL()

void recomputeOrientation()

{

x = cameraRadius \* sinf(cameraTheta)\*sinf(cameraPhi);

z = cameraRadius \* cosf(cameraTheta)\*sinf(cameraPhi);

y = cameraRadius \* cosf(cameraPhi);

glutPostRedisplay();

}

void mouseCallback(int button, int state, int thisX, int thisY)

{

//update the left and right mouse button states, if applicable

if (button == GLUT\_LEFT\_BUTTON)

leftMouseButton = state;

else if (button == GLUT\_RIGHT\_BUTTON)

rightMouseButton = state;

//and update the last seen X and Y coordinates of the mouse

mouseX = thisX;

mouseY = thisY;

}

void mouseMotion(int x, int y)

{

if (leftMouseButton == GLUT\_DOWN)

{

cameraTheta += (mouseX - x)\*0.005;

cameraPhi += (mouseY - y)\*0.005;

// make sure that phi stays within the range (0, PI)

if (cameraPhi <= 0)

cameraPhi = 0 + 0.001;

if (cameraPhi >= PI)

cameraPhi = PI - 0.001;

recomputeOrientation(); //update camera (x,y,z)

}

// camera zoom in/out

else if (rightMouseButton == GLUT\_DOWN) {

double totalChangeSq = (x - mouseX) + (y - mouseY);

cameraRadius += totalChangeSq \* 0.01;

//limit the camera radius to some reasonable values so the user can't get lost

if (cameraRadius < 2.0)

cameraRadius = 2.0;

if (cameraRadius > 10.0)

cameraRadius = 10.0;

recomputeOrientation(); //update camera (x,y,z) based on (radius,theta,phi)

}

mouseX = x;

mouseY = y;

}

void display() {

// Draws the scene, consisting of three teapots with different textures.

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glEnable(GL\_TEXTURE\_2D);

glTexGeni(GL\_S, GL\_TEXTURE\_GEN\_MODE, GL\_OBJECT\_LINEAR);

glTexGeni(GL\_T, GL\_TEXTURE\_GEN\_MODE, GL\_OBJECT\_LINEAR);

glEnable(GL\_TEXTURE\_GEN\_S);

glEnable(GL\_TEXTURE\_GEN\_T);

/\* Draw a teapot with the second texture at (0,-0.5,0) \*/

glBindTexture(GL\_TEXTURE\_2D, texID[0]); // Bind texture #1 for use on the secondteapot.

glPushMatrix();

gluLookAt(x, y, z, //camera is located at (x,y,z)

0, 0, 0, //camera is looking at (0,0,0)

0.0f, 1.0f, 0.0f);

glutSolidCube(1);

glPopMatrix();

// Draw a teapot with the third texture at (2.3,0.3,0)

// enable automatic texture coordinate generation

// suppose that the texture has been loaded to texImage

glBindTexture(GL\_TEXTURE\_2D, texID[1]);

glPushMatrix();

glTranslatef(-2.5, 0, 0);

glutSolidTeapot(1); // draw a teapot with this sphere map

glBindTexture(GL\_TEXTURE\_2D, texID[1]);

glTranslatef(5, 0, 0.0);

glutSolidSphere(1.0, 20, 20); // draw a sphere with this sphere map

glPopMatrix();

glDisable(GL\_TEXTURE\_GEN\_S);

glDisable(GL\_TEXTURE\_GEN\_T);

glutSwapBuffers(); // (required for double-buffered drawing)

}

// --------------- keyboard event function ---------------------------------------

void specialKeyFunction(int key, int x, int y) {

// Change rotation amounts in response to arrow and home keys.

if (key == GLUT\_KEY\_LEFT)

rotateY -= 15;

else if (key == GLUT\_KEY\_RIGHT)

rotateY += 15;

else if (key == GLUT\_KEY\_DOWN)

rotateX += 15;

else if (key == GLUT\_KEY\_UP)

rotateX -= 15;

else if (key == GLUT\_KEY\_HOME)

rotateX = rotateY = 0;

glutPostRedisplay();

}

// --------------------- texture loading -----------------------------------------

/\* This function loads three textures from texture files, which must be available

\* to the program when it is run. Paths to the files are stored in the global

\* array textureFileNames. The function generates three texture object identifiers

\* and stores them in the global array textID so that they can be used when

\* binding textures in display(). It then loads the three texture images into

\* the texture objects. It calles glTexParameteri for each texture to change

\* the minification filter to GL\_LINEAR (without this, the texture won't work

\* because there are no mipmaps for the textures).

\*/

void loadTextures() {

int i;

glGenTextures(3, texID); // Get the texture object IDs.

for (i = 0; i < 3; i++) {

void\* imgData; // Pointer to image color data read from the file.

int imgWidth; // The width of the image that was read.

int imgHeight; // The height.

FREE\_IMAGE\_FORMAT format = FreeImage\_GetFIFFromFilename(textureFileNames[i]);

if (format == FIF\_UNKNOWN) {

printf("Unknown file type for texture image file %s\n",

textureFileNames[i]);

continue;

}

FIBITMAP\* bitmap = FreeImage\_Load(format, textureFileNames[i], 0); // Read image from file.

if (!bitmap) {

printf("Failed to load image %s\n", textureFileNames[i]);

continue;

}

FIBITMAP\* bitmap2 = FreeImage\_ConvertTo24Bits(bitmap); // Convert to RGB or BGR format

FreeImage\_Unload(bitmap);

imgData = FreeImage\_GetBits(bitmap2); // Grab the data we need from thebitmap.

imgWidth = FreeImage\_GetWidth(bitmap2);

imgHeight = FreeImage\_GetHeight(bitmap2);

if (imgData) {

printf("Texture image loaded from file %s, size %dx%d\n",

textureFileNames[i], imgWidth, imgHeight);

glBindTexture(GL\_TEXTURE\_2D, texID[i]); // Will load image data into texture object #i

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, imgWidth, imgHeight, 0,

GL\_BGR\_EXT,

GL\_UNSIGNED\_BYTE, imgData);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR); //Required since there are no mipmaps.

}

else {

printf("Failed to get texture data from %s\n", textureFileNames[i]);

}

}

}

// ----------------- main routine -------------------------------------------------

int main(int argc, char\*\* argv) {

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_DEPTH);

glutInitWindowSize(750, 300);

glutInitWindowPosition(100, 100);

glutCreateWindow("Bryce Chalfant Assignment 5");

initGL();

loadTextures();

cameraRadius = 2.0f;

cameraTheta = 1;

cameraPhi = 1;

recomputeOrientation();

glutDisplayFunc(display);

glutSpecialFunc(specialKeyFunction);

glutMouseFunc(mouseCallback);

glutMotionFunc(mouseMotion);

glutMainLoop();

return 0;

}