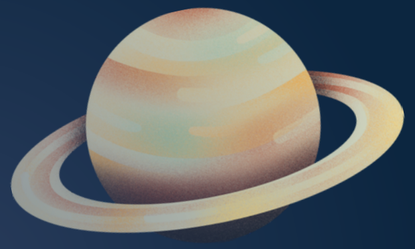
OPENGL SOLAR SYSTEM PROJECT



Dana AL-Duayji
Emtenan Al-Fozan
Shrouq AL-qaied
Deema AL-Freidy

GENRAL OVERVIEW
-MAIN FUNCTION
-INIT_LOADALLTEXTURE FUNCTION

let's dive into the details of the
Solar System program!

Overview of the whole program

```c
#include <GL/freeglut.h>
#include <math.h>
#include <stdio.h>

// Constants for window size and distances
#define Windowwidth  1024
#define Windowheight 1024
#define Distance 100000000
#define Eradius 16000000
const GLfloat Pi = 3.1415926536f;

// Global variables
static int day = 0;
GLdouble Angle = 80.0;
GLdouble aix_x = 0.0, aix_y = 2000000000, aix_z = 2000000000;

// Texture IDs for celestial bodies
GLuint tbg, tsun, tearth, tmercu, tven, tmars, tjup, tsat, tura, tnep;

// Function to check if a number is a power of two
int power_of_two(int n) {
    if (n <= 0)
        return 0;
    return (n & (n - 1)) == 0;
}

// Function to load a texture from a file
GLuint LoadTexture(const char* filename) {
    // ... (Code for loading texture)
}

// Initialize and load all texture files
void init_LoadallTexture() {
    tsun = LoadTexture("pictures/sol.bmp");
    tbg = LoadTexture("pictures/bg.bmp");
    tearth = LoadTexture("pictures/terra.bmp");
    tmercu = LoadTexture("pictures/mercurio.bmp");
    tven = LoadTexture("pictures/venus.bmp");
    tmars = LoadTexture("pictures/marte.bmp");
    tjup = LoadTexture("pictures/jupiter.bmp");
    tsat = LoadTexture("pictures/saturno.bmp");
    tura = LoadTexture("pictures/urano.bmp");
    tnep = LoadTexture("pictures/neptuno.bmp");
```

```c
// Function to draw the background
void get_bg() {
    // ... (Code for setting up background)
}

// Function to draw an orbit
void drawOrbit(double radius, int numVertices = 100) {
    // ... (Code for drawing orbit)
}

// Function to draw the sun
void get_sun() {
    // ... (Code for drawing sun)
}

// Functions to draw each planet (e.g., get_mercu, get_ven, etc.)
// ... (Code for drawing planets)

// Display Function
void myDisplay(void) {
    // ... (Code for setting up display)
}

// Timer function to control the frame rate
void timerProc(int id) {
    ++day;
    glutPostRedisplay();
    glutTimerFunc(50, timerProc, 1);
}

// Keyboard function for user interaction
void mykeyboard(unsigned char key, int x, int y) {
    // ... (Code for handling keyboard input)
}

// Main function
int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(Windowwidth, Windowheight);

    glutCreateWindow("Solar System");
    init_LoadallTexture();
    glutDisplayFunc(&myDisplay);
    glutKeyboardFunc(mykeyboard);
    glutTimerFunc(50, timerProc, 1);
    glutMainLoop();

    return 0;
}
```

Overview of Loadad Texture function

function responsible for loading texture data from an image file and creating an OpenGL texture

```
GLuint LoadTexture(const char* filename)
{

    GLint width, height, total_bytes;
    GLubyte* pixels = 0;
    GLuint  texture_ID = 0;


    FILE* pFile = fopen(filename, "rb");
    if (pFile == 0)
        return 0;

    fseek(pFile, 0x0012, SEEK_SET);
    fread(&width, 4, 1, pFile);
    fread(&height, 4, 1, pFile);
    fseek(pFile, 54, SEEK_SET);

    {
        GLint line_bytes = width * 3;
        while (line_bytes % 4 != 0)
            ++line_bytes;
        total_bytes = line_bytes * height;
    }


    pixels = (GLubyte*)malloc(total_bytes);
    if (pixels == 0)
    {
        fclose(pFile);
        return 0;
    }

    if (fread(pixels, total_bytes, 1, pFile) <= 0)
    {
        free(pixels);
        fclose(pFile);
        return 0;
```

FILE OPENING:
OPENS THE SPECIFIED FILE IN BINARY MODE ("RB"). IF THE FILE CANNOT BE OPENED, IT RETURNS 0, INDICATING A FAILURE IN LOADING THE TEXTURE.

READING IMAGE DIMENSIONS:
READS THE WIDTH AND HEIGHT VALUES FROM THE FILE HEADER.

CALCULATING THE TOTAL NUMBER OF BYTES REQUIRED TO STORE AN IMAGE

MEMORY ALLOCATION:
ALLOCATES MEMORY TO STORE THE PIXEL DATA OF THE IMAGE. IF THE ALLOCATION FAILS, IT RETURNS 0

READING PIXEL DATA:
READS THE PIXEL DATA FROM THE FILE INTO THE ALLOCATED MEMORY. IF THE READ OPERATION FAILS, IT FREES THE ALLOCATED MEMORY AND RETURNS 0.

```
    glGenTextures(1, &texture_ID);
    if (texture_ID == 0)
    {
        free(pixels);
        fclose(pFile);
        return 0;
    }

    glBindTexture(GL_TEXTURE_2D, texture_ID);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0,
        GL_BGR_EXT, GL_UNSIGNED_BYTE, pixels);
    free(pixels);
    return texture_ID;
}

//Initilize to load files
void init_LoadallTexture()
{
    tsun = LoadTexture("pictures/sol.bmp");
    tbg = LoadTexture("pictures/bg.bmp");
    tearth = LoadTexture("pictures/terra.bmp");
    tmercu = LoadTexture("pictures/mercurio.bmp");
    tven = LoadTexture("pictures/venus.bmp");
    tmars = LoadTexture("pictures/marte.bmp");
    tjup = LoadTexture("pictures/jupiter.bmp");
    tsat = LoadTexture("pictures/saturno.bmp");
    tura = LoadTexture("pictures/urano.bmp");
    tnep = LoadTexture("pictures/neptuno.bmp");
}
```

TEXTURE GENERATION:
GENERATES AN OPENGL TEXTURE ID. IF THE GENERATION FAILS, IT FREES THE ALLOCATED MEMORY AND RETURNS 0.

SETTING TEXTURE PARAMETERS:
BINDS THE TEXTURE AND SETS VARIOUS PARAMETERS SUCH AS FILTERING AND WRAPPING MODES..

LOADING TEXTURE DATA INTO OPENGL

INIT_LOADALLTEXTURE FUNCTION INITIALIZES GLOBAL VARIABLES WITH TEXTURE IDS

THE MYDISPLAY FUNCTION RESPONSIBLE FOR RENDERING THE SOLAR SYSTEM.

let's dive into the details of the myDisplay function and how it contributes to rendering the solar system using OpenGL

THIS FUNCTION IS THE CORE OF GRAPHICS RENDERING. IT SETS UP THE PROJECTION, VIEW, DRAWS ORBITS, AND CALLS FUNCTIONS TO RENDER EACH PLANET. IT ALSO HANDLES SWAPPING BUFFERS FOR DOUBLE BUFFERING.

```
void myDisplay(void)

{

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    get_bg();
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(Angle, 1, 1, 10000000000);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(aix_x, aix_y, aix_z, 0, 0, 0, 0, 0, 1);

    // Draw orbits
    glColor3f(0.5f, 0.5f, 0.5f);
    drawOrbit(Distance);
    drawOrbit(2 * Distance);
    drawOrbit(3 * Distance);
    drawOrbit(4 * Distance);
    drawOrbit(6.5 * Distance);
    drawOrbit(10.5 * Distance);
    drawOrbit(13 * Distance);
    drawOrbit(15 * Distance);

    get_sun();
    get_mercu();
    get_ven();
    get_earth();
    get_mars();
    get_jup();
    get_sat();
    get_ura();
    get_nep();

    glFlush();
    glutSwapBuffers();

}
```

CLEARS THE COLOR AND DEPTH BUFFERS TO PREPARE FOR RENDERING.

CALLS THE GET_BG FUNCTION TO SET UP THE BACKGROUND USING “GLCLEARCOLOR” AND “GLCLEAR” TO CREATE A CLEAR CANVAS FOR RENDERING.

GLMATRIXMODE(GL_PROJECTION):

SETS THE CURRENT MATRIX MODE TO THE PROJECTION MATRIX. THE PROJECTION MATRIX IS RESPONSIBLE FOR TRANSFORMING 3D COORDINATES INTO 2D COORDINATES FOR RENDERING ON THE SCREEN.

GLLOADIDENTITY():

LOADS THE IDENTITY MATRIX INTO THE CURRENT MATRIX. THIS RESETS ANY PREVIOUS TRANSFORMATIONS APPLIED TO THE MATRIX.

GLUPERSPECTIVE(ANGLE, 1, 1, 10000000000):

CONFIGURES A PERSPECTIVE PROJECTION MATRIX.

- ANGLE: THE FIELD OF VIEW ANGLE IN DEGREES.
- 1: THE ASPECT RATIO, WHICH IS USUALLY SET TO 1 FOR A SQUARE VIEWPORT.
- 1: THE NEAR CLIPPING PLANE DISTANCE.
- 10000000000: THE FAR CLIPPING PLANE DISTANCE. THIS LARGE VALUE MEANS THE FAR CLIPPING PLANE IS AT A CONSIDERABLE DISTANCE, SIMULATING AN INFINITE VIEW.

SETS THE COLOR FOR ORBITS AND CALLS THE DRAWORBIT FUNCTION TO DRAW CIRCULAR PATHS FOR EACH BODY

GLMATRIXMODE(GL_MODELVIEW):

SETS THE CURRENT MATRIX MODE TO THE MODELVIEW MATRIX. THE MODELVIEW MATRIX IS RESPONSIBLE FOR TRANSFORMING THE MODEL'S VERTICES FROM WORLD SPACE TO VIEW SPACE.

GLULOOKAT(AIX_X, AIX_Y, AIX_Z, 0, 0, 0, 0, 0, 1):

DEFINES THE CAMERA POSITION AND ORIENTATION.

- **(AIX_X, AIX_Y, AIX_Z)**: SPECIFIES THE CAMERA'S POSITION IN WORLD COORDINATES.
- **(0, 0, 0)**: SPECIFIES THE POINT THE CAMERA IS LOOKING AT.
- **(0, 0, 1)**: SPECIFIES THE UP VECTOR, WHICH DETERMINES THE ORIENTATION OF THE CAMERA.

CALLS FUNCTIONS TO RENDER EACH PLANET. THESE FUNCTIONS HANDLE THE RENDERING OF EACH ONE. THEY SET UP LIGHTING, MATERIALS, AND TEXTURES FOR EACH BODY AND USE OPENGL FUNCTIONS LIKE GLPUSHMATRIX, GLROTATEF, GLTRANSLATE AND GLUSPHERE TO POSITION, ROTATE, AND RENDER THE TEXTURED SPHERES.

THE TIMERPROC FUNCTION PLAYS A CRUCIAL ROLE IN CONTROLLING THE ANIMATION AND UPDATING THE DAY COUNT IN THE SOLAR SYSTEM SIMULATION.

let's dive into the details of the TimerProc function and how it contributes to animate the solar system using OpenGL

- FUNCTION CALLS GLUTPOSTREDISPLAY(). THIS TRIGGERS A REDRAW OF THE SCENE, UPDATING THE POSITIONS OF THE CELESTIAL BODIES BASED ON THE INCREMENTED DAY.
- THIS CONTINUOUS TRIGGERING OF REDRAWS CREATES THE ILLUSION OF MOTION IN THE SIMULATION, MAKING THE PLANETS ORBIT THE SUN

```
void timerProc(int id)
{
    ++day;
    glutPostRedisplay();
    glutTimerFunc(50, timerProc, 1);
}
```

THE FUNCTION IS CALLED AT REGULAR INTERVALS USING GLUTTIMERFUNC.

IT INCREMENTS THE DAY VARIABLE, SIMULATING THE PASSAGE OF TIME IN THE ANIMATION. THIS VARIABLE DRIVES THE ROTATION OF PLANETS AROUND THE SUN.

```
int main(int argc, char* argv[])

{


    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(Windowwidth, Windowheight);

    glutCreateWindow("Solar System");
    init_LoadallTexture();
    glutDisplayFunc(&myDisplay);
    glutKeyboardFunc(mykeyboard);
    glutTimerFunc(50, timerProc, 1);
    glutMainLoop();

    return 0;


}
```

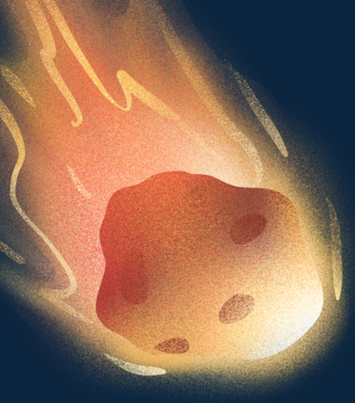
GLUTTIMERFUNC:

IS A FUNCTION PROVIDED BY THE GLUT (OPENGL UTILITY TOOLKIT) LIBRARY THAT ENABLES THE SCHEDULING OF RECURRING TIMER EVENTS.

THIS LINE SETS UP A TIMER THAT CALLS THE TIMERPROC FUNCTION EVERY 50 MILLISECONDS.
INSIDE TIMERPROC, THE DAY VARIABLE IS INCREMENTED, SIMULATING THE PASSAGE OF DAYS IN THE SOLAR SYSTEM SIMULATION.

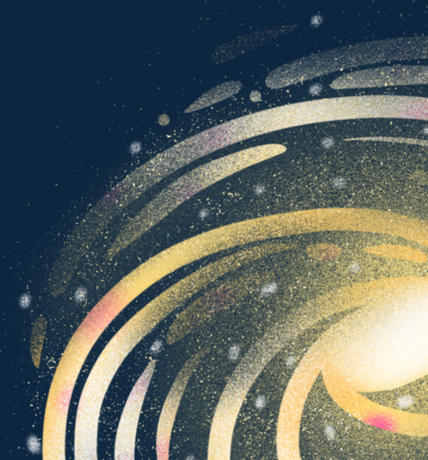
ANIMATION CONTROL:

- THE TIMERPROC FUNCTION IS CALLED AT REGULAR INTERVALS USING GLUTTIMERFUNC.
- IT INCREMENTS THE DAY VARIABLE, SIMULATING THE PASSAGE OF TIME IN THE ANIMATION. THIS VARIABLE DRIVES THE ROTATION OF PLANETS AROUND THE SUN.



THE MYKEYBOARD FUNCTION HANDLES USER KEYBOARD INPUT TO MANIPULATE THE VIEW WITHIN THE SOLAR SYSTEM SIMULATION.

let's dive into the details of the mykeyboard function and how it contributes to change the view of the solar system using opengl



REPRESENTS THE ASCII VALUE OF THE PRESSED KEY

MOUSE COORDINATES AT THE TIME OF THE KEY PRESS

```
void mykeyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 'W':
    case 'w':
        aix_y += 100000000;
        aix_z -= 100000000;
        break;
    case 'S':
    case 's':
        aix_y -= 100000000;
        aix_z += 100000000;
        break;
    case 'A':
    case 'a':
        aix_x -= 100000000;
        aix_z += 100000000;
        break;
    case 'D':
    case 'd':
        aix_x += 100000000;
        aix_z -= 100000000;
        break;


    case 'J':
    case 'j':
        Angle -= 5.0;
        break;
    case 'L':
    case 'l':
        Angle += 5.0;
        break;
    }
    glutPostRedisplay();

}
```

W OR W: MOVES THE VIEW UPWARD AND SHIFTS THE CAMERA POSITION.

S OR S: MOVES THE VIEW DOWNWARD AND ADJUSTS THE CAMERA.

A OR A: MOVES THE VIEW LEFTWARD.

D OR D: MOVES THE VIEW RIGHTWARD

J OR J: ROTATES THE SCENE/CHANGES THE ANGLE TO THE LEFT.

L OR L: ROTATES THE SCENE/CHANGES THE ANGLE TO THE RIGHT.

AFTER PROCESSING USER INPUT, GLUTPOSTREDISPLAY() IS CALLED TO UPDATE THE DISPLAY ACCORDING TO THE MODIFIED VIEW OR ANGLE.

BY CALLING GLUTKEYBOARDFUNC AND PASSING MYKEYBOARD AS AN ARGUMENT, YOU'RE ESSENTIALLY TELLING GLUT, "WHENEVER A KEY IS PRESSED, CALL MYKEYBOARD TO HANDLE THAT EVENT."

```
int main(int argc, char* argv[])

{


    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(Windowwidth, Windowheight);

    glutCreateWindow("Solar System");
    init_LoadallTexture();
    glutDisplayFunc(&myDisplay);
    glutKeyboardFunc(mykeyboard);
    glutTimerFunc(50, timerProc, 1);
    glutMainLoop();

    return 0;


}
```

GLUTKEYBOARDFUNC IS USED TO SPECIFY A FUNCTION (MYKEYBOARD IN THIS CASE) THAT WILL BE CALLED WHENEVER A KEYBOARD KEY IS PRESSED.


THANK YOU!