RV College of Engineering®, BENGALURU - 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

Department of Computer Science and Engineering

"Tetris Game using OpenGL"

COMPUTER GRAPHICS LAB (16CS73)
OPEN ENDED EXPERIMENT REPORT

VII SEMESTER
2020-2021

Submitted by

Sahana Srinivasan 1RV17CS132

Ruchita R. Biradar 1RV17CS126

Under the Guidance of

Prof. Mamatha T
Department of CSE, R.V.C.E.,
Bengaluru - 560059

Abstract

The aim of the project is to make use of OpenGL libraries to develop an application for a high-resolution version of the popular game, Tetris. The game Tetris has always been a household favourite, right from being played on video game consoles, to being introduced on television screens. Right from being smuggled out of the Soviet Union and onto our computers, Tetris has come a long way the past 36 years. Mathematicians have written theses on this game, and video game developers have fought for rights over it at Las Vegas Conventions, all this goes to show just how beautiful an example Tetris is, for the world of graphics to begin its revolution. Nevertheless, it goes without saying, that with so many versions of the game now available, it is only fair to make an OpenGL version of Tetris to do justice to it.

RV College of Engineering®, BENGALURU - 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

Department of Computer Science and Engineering

CERTIFICATE

Certified that the **Open-Ended Experiment** titled "Tetris Game using OpenGL" has been carried out by **Sahana Srinivasan(1RV17CS132) and Ruchita R. Biradar (1RV17CS126),** bonafide students of RV College of Engineering, Bengaluru, have submitted in partial fulfillment for the **Internal Assessment of Course: COMPUTER GRAPHICS LAB (16CS73)** during the year 2020-2021. It is certified that all corrections/suggestions indicated for the internal Assessment have been incorporated in the report.

Prof. Mamatha T

Faculty Incharge,
Department of CSE,
R.V.C.E., Bengaluru –59

Dr. Ramakanth Kumar P

Head of Department
Department of CSE
R.V.C.E., Bengaluru–59

RV College of Engineering® , BENGALURU - 560059
(Autonomous Institution Affiliated to VTU)

Department of Computer Science and Engineering

DECLARATION

We, **Sahana Srinivasan(1RV17CS132) and Ruchita R. Biradar (1RV17CS126)** the students of Seventh Semester B.E., Computer Science and Engineering, R.V. College of Engineering, Bengaluru hereby declare that the mini-project titled **"Tetris Game using OpenGL"** has been carried out by us and submitted in partial fulfillment for the **Internal Assessment of Course: COMPUTER GRAPHICS LAB (16CS73) - Open-Ended Experiment** during the year 2020-2021. We do declare that matter embodied in this report has not been submitted to any other university or institution for the award of any other degree or diploma.

Place: Bengaluru **Sahana Srinivasan**
Date: 18 December 2020 **Ruchita R. Biradar**

TABLE OF CONTENTS

1. Introducion

1.1 Computer Graphics

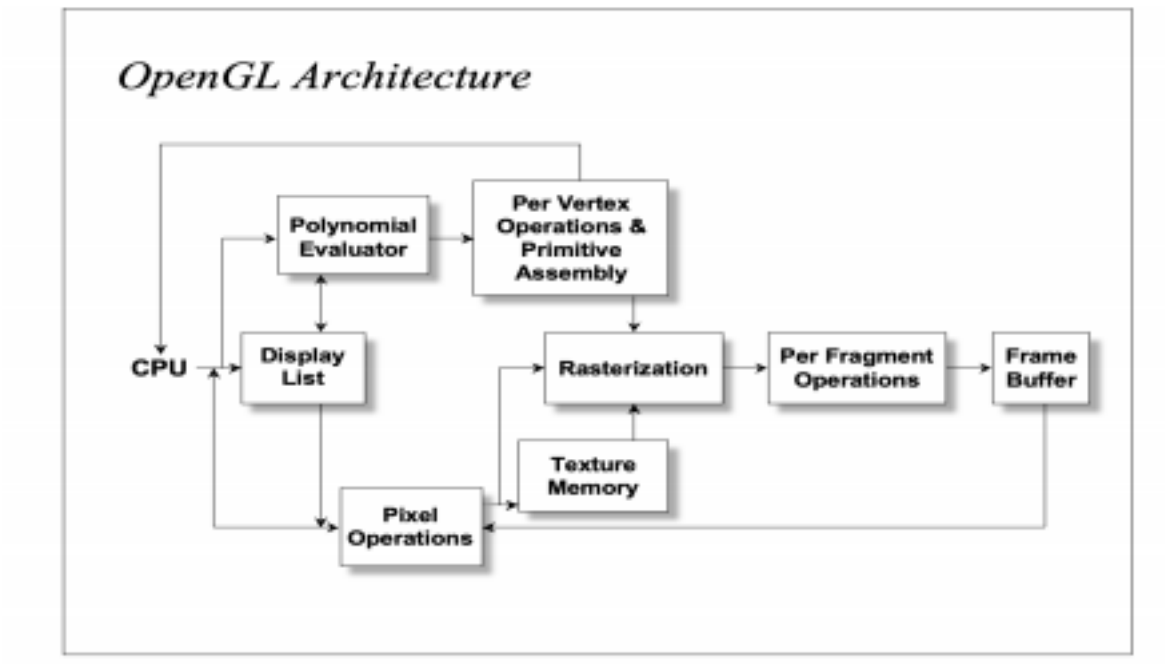
Computer graphics is the discipline of generating images with the aid of computers. Today, computer graphics is a core technology in digital photography, film, video games, cell phone and computer displays, and many specialized applications. A great deal of specialized hardware and software has been developed, with the displays of most devices being driven by the computer graphics hardware. It is a vast and recently developed area of computer science. The phrase was coined in 1960 by computer graphics researchers Verne Hudson and William Fetter of Boeing. It is often abbreviated as CG, or typically in the context of the film as CGI.

Some topics in computer graphics include user interface design, sprite graphics, rendering, ray tracing, geometry processing, computer animation, vector graphics, 3D modelling, shaders, GPU design, implicit surface visualization, image processing, computational photography, scientific visualization, computational geometry and computer vision, among others. The overall methodology depends heavily on the underlying sciences of geometry, optics, physics, and perception.

Computer graphics is responsible for displaying art and image data effectively and meaningfully to the consumer. It is also used for processing image data received from the physical world. Computer graphics development has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, and graphic design in general.

1.2 OpenGL

1.2.1 OpenGL Graphics Architecture



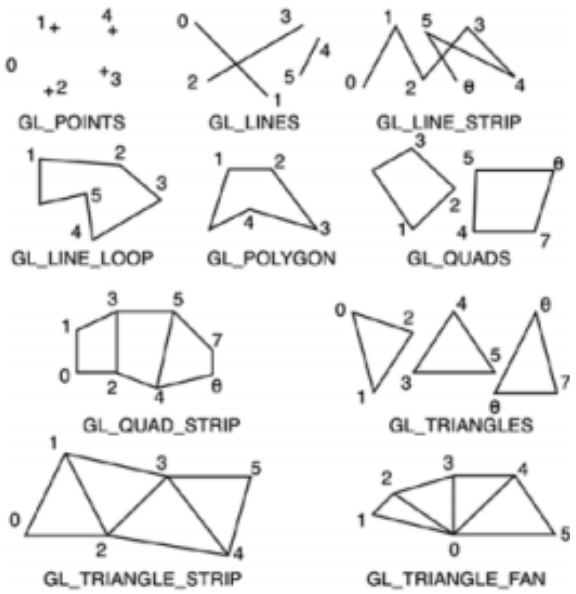
The architecture of OpenGL is based on a client-server model. An application program written to use the OpenGL API is the "client" and runs on the CPU. The implementation of the OpenGL graphics engine is the "server" and runs on the GPU.
Geometry and many other types of attributes are stored in buffers called Vertx Buffer Objects. These buffers are allocated on the GPU and filled by your CPU program.

Modelling, rendering, and interaction is very much a cooperative process between the CPU client program and the GPU server programs written in GLSL. An important part of the design process is to decide how best to divide the work and how best to package and communicate the required information from the CPU to the GPU.

1.2.2 Primitives and Attributes

OpenGL supports several basic primitive types, including points, lines, quadrilaterals, and general polygons. All of these primitives are specified using a sequence of vertices. The diagram below shows the basic primitive types, where the numbers indicate the order in which the vertices have been specified. Note that for the GL_LINES primitive only every second vertex causes a line segment to be drawn.
Similarly, for the GL_TRIANGLES primitive, every third vertex causes a triangle to be

drawn. Note that for the GL_TRIANGLE_STRIP and GL_TRIANGLE_FAN primitives, a new triangle is produced for every additional vertex. All of the closed primitives shown below are solid-filled, with the exception of GL_LINE_LOOP, which only draws lines connecting the vertices.

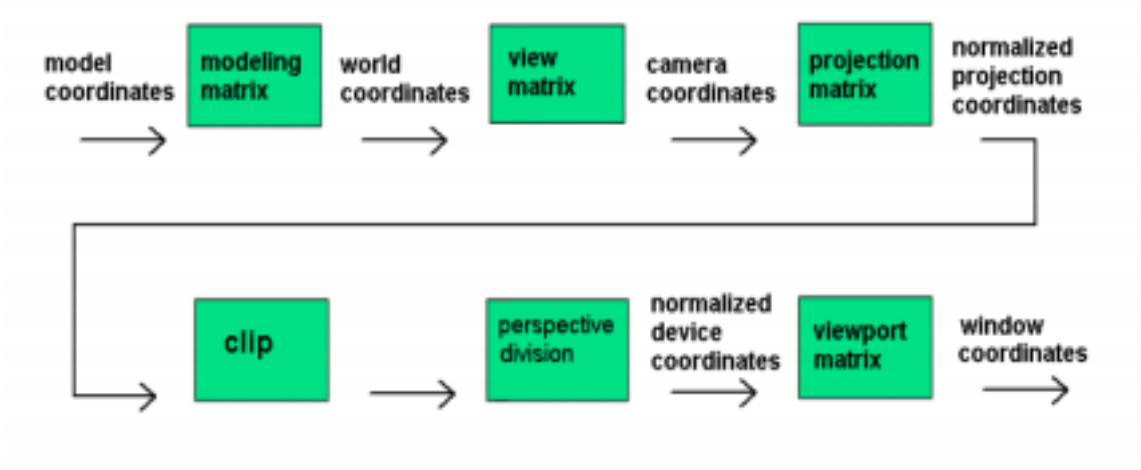


In general, a parameter that affects the way a primitive is to be displayed is referred to as an attribute parameter. Some attribute parameters, such as colour and size, determine the fundamental characteristics of a primitive. Other attributes specify how the primitive is to be displayed under special conditions. Text can appear reading from left to right, slanted diagonally across the screen, or in vertical columns. Individual characters can be displayed in different fonts, colours, and sizes. And we can apply intensity variations at the edges of objects to smooth out the raster stair-step effect.

A graphics system that maintains a list for the current values of attributes and other parameters is referred to as a state system or state machine. Attributes of output primitives and some other parameters, such as the current frame-buffer position, are referred to as state variables or state parameters. When we assign a value to one or more state parameters, we put the system into a particular state. And that state remains in effect until we change the value of a state parameter.

1.2.3 Colour and Viewing

OpenGL maintains a current drawing colour as part of its state information. The glColor() function calls are used to change the current drawing colour. assigned using the glColor function call. Like glVertex(), this function exists in various instantiations. Colour components are specified in the order of red, green, blue. Colour component values are in the range [0...1], where 1 corresponds to maximum intensity. For unsigned bytes, the range corresponds to [0...255]. All primitives following the fragment of code given below would be drawn in green, assuming no additional glColor() function calls are used.



The coordinates we specify using the glVertex* commands are the model coordinates. TheglRotate, glTranslate and glScale commands are used to transform the model into the desired orientation and size. After applying the modelling transformations to the model coordinates what we get are world coordinates. The Modelling transformations give rise to 4×4 matrices.

OpenGL Control Functions

- Window – A rectangular area of our display.
- glutInit allows the application to get command line arguments and initializes the system

  glutInit(int *argc, char **argv) initializes GLUT and processes any command-line arguments (for X, this would be options like -display and -geometry).
- gluInitDisplayMode requests properties for the window (the rendering context)

- RGB colour
- Single buffering
- Properties logically ORed together

glutInitDisplayMode(unsigned int mode) specifies whether to use an RGBA or color index color model. You can also specify whether you want a single- or double-buffered window.

glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH). If you want a window with double buffering, the RGBA colour model, and a depth buffer, you might call

- glutWindowSize(int width, int size) in pixels
- glutWindowPosition(int x, int y) specifies the screen location for the upper-left corner of your window
- glutCreateWindow(char *string) create a window with a particular title

1.3 Proposed System

The system is a 2-dimensional Tetris game that is designed to be played with the keyboard's arrow keys. Tetris is a tile-matching puzzle video game originally designed and programmed by Soviet Russian software engineer Alexey Pajitnov. The first playable version was completed on June 6, 1984, while he was working for the Dorodnitsyn Computing Centre of the Academy of Science of the Soviet Union in Moscow. He derived its name from combining the Greek numerical prefix tetra- (the falling pieces contain 4 segments) and tennis, Pajitnov's favourite sport. The name is also used in-game to refer to the play where four lines are cleared at once. We will be implementing this classic game that was the first game to be exported from the Soviet Union to the United States of America, with the help of OpenGL control functions and primitives.

1.3.1 Objective of the project

The project is designed to run in a single window on the MacOS operating system. To comply with the constraints on the platform, the project uses the Xcode environment for development and uses the OpenGL and GLUT frameworks as packages. With respect to the design, the blocks or tiles that are in the game are designed to be of

different colours and their shapes are randomly chosen from a previously defined set of allowable shapes. The blocks also perfectly match the grid cells. The blocks can be rotated when they are descending in air and the objective of the game is to align them in the bottom most row in the grid. In a tetris game, upon completion of the lowest row with blocks, the row disappears and every block is moved one row down. The speed of the descending blocks increases with time and the game gets progressively harder. The aim of our project is to simulate the game with all these features in OpenGL.

1.3.2 Methodology

There are 3 main elements to the program files: Game, Tetromino and the Board. Each of these 3 elements are defined as classes with corresponding methods and private and public variables.

- Game class manages the interaction between the user, each tetromino block and the board.
- Tetromino class is responsible for defining the shapes and the colors of each block and rotating functions.
- Board class maintains a matrix to represent each cell in the grid where the game is being played.

All these elements work in unison and enable all the functionality in the Tetris game. These elements also use helper libraries and packages for matrix and vector calculations which are bundled together in the Angel file in the include folder.

1.3.3 Scope

It is the basic implementation of the classic game Tetris. It is created by using minimum classes and functions. OpenGL has been used effectively for all the animation without the use of any additional libraries. This allows for the creation and implementation of many other games using only OpenGL.

2. Requirement Specifications

2.1. Hardware Requirements

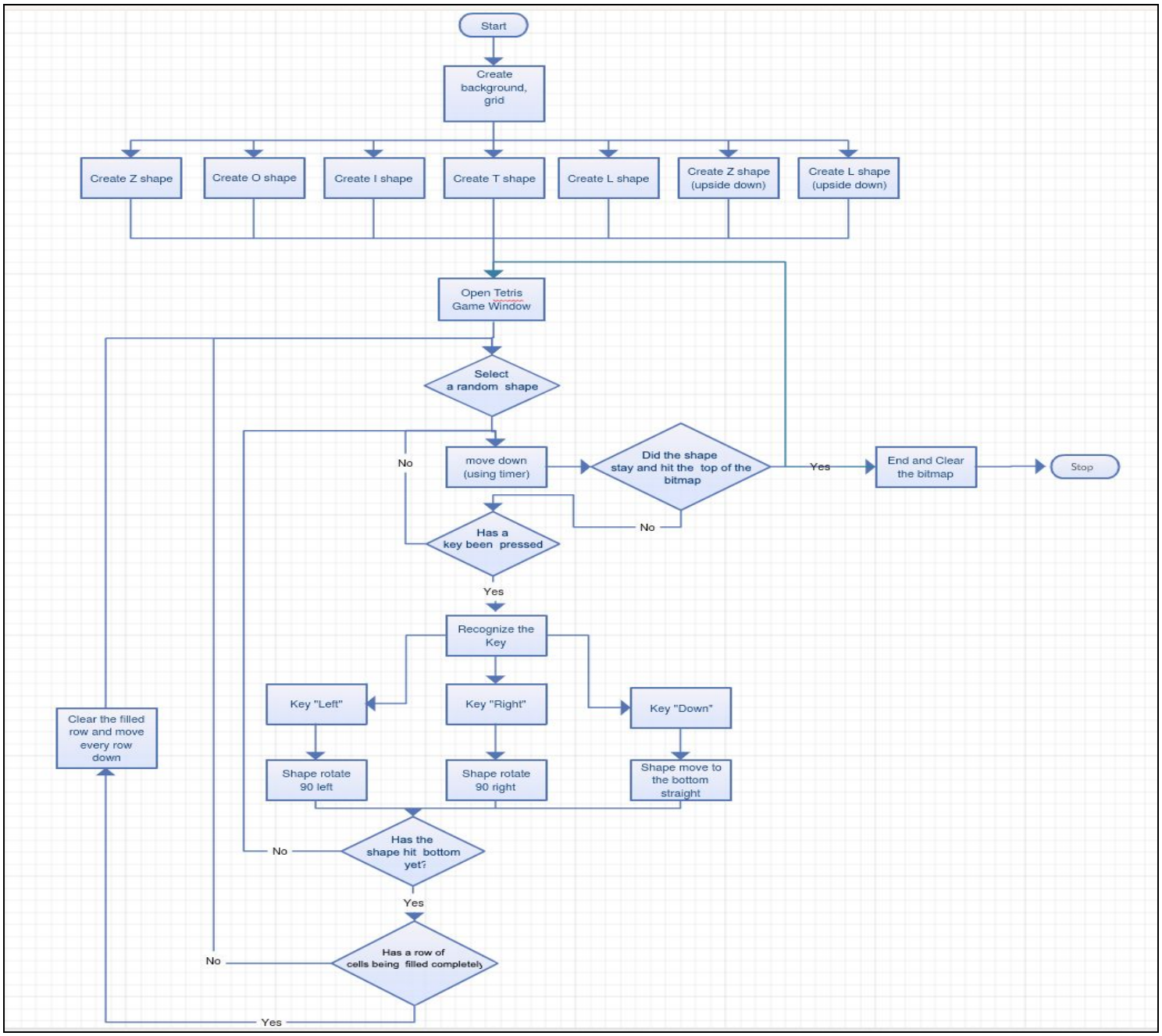
- Monitor
- Keyboard
- Mouse
- CPU

2.2 Software Requirements

- Linux or MacOS
- OpenGL
- Xcode
- GCC
- G++
- OpenGL Utility Toolkit

3. System Design and Implementation

3.1 Flow Diagram



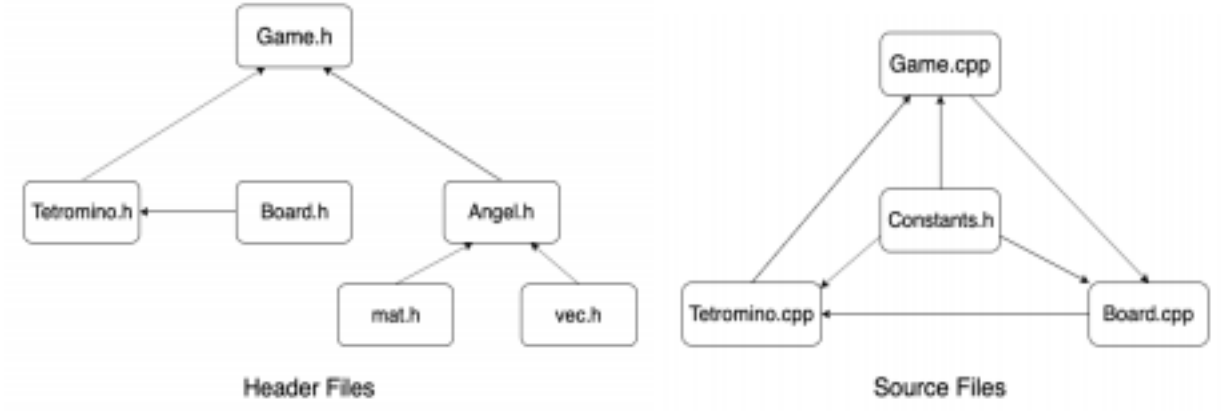
3.2 Modular Description

Three modules in the project are:

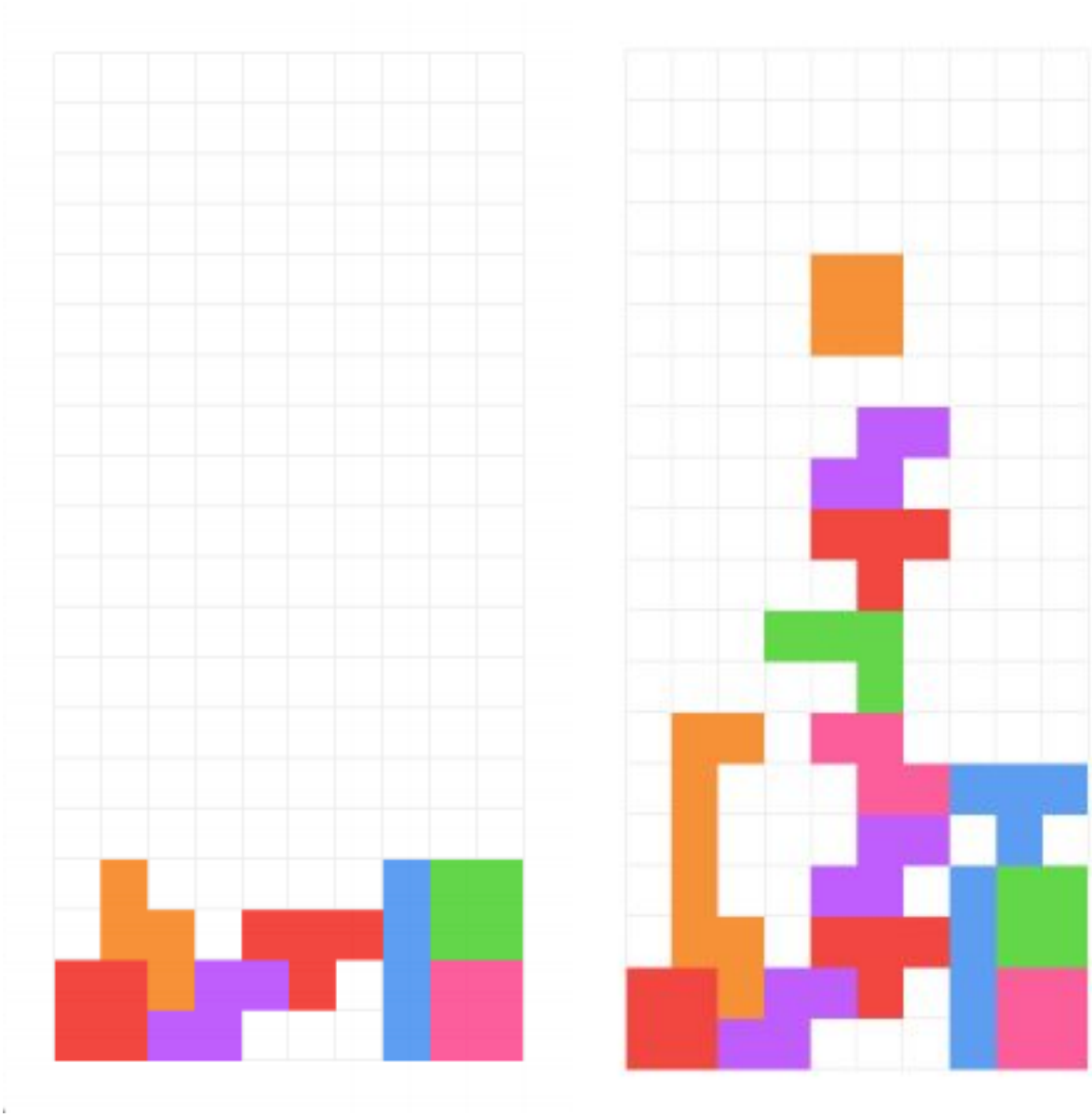
- Game
- Tetromino
- Board

The Game class manages the score, the collisions and the running of the OpenGL functions to create the window and make sure the game is functional and interactive. It also defines the necessary functions to identify and intercept the keyboard to control the tetrominos.

The Tetromino class defines each block that falls down in the window. The shapes, colors, orientations and the randomization of these attributes is defined in this class. The rotation functions for each tetromino are also defined in this class. When the game class calls for an instance of tetromino it has to call the constructor of this class to get an object on which it executes the relevant functions. The reset function on the tetromino also puts the game back in initial state and starts dropping the new tetrominos from a fixed position with a fixed initial speed.
The Board class maintains the state of each cell in the board. The board is a 10x20 vertical white colored grid. A matrix keeping track of each cell is present and initially every cell will have a value of -1. As and when the tetrominos come and settle on the bottom rows, the respective cells in this matrix are updated and on completion of a row, the score is updated as well. This class takes care of collision detection and detecting if the top of the board has been reached at any point in the game.



4. Results and Snapshots



The different stages in a game showcasing the different colored tetrominos. The speed at which these blocks descend increases over time.

5. Conclusion

This project is the basic implementation of the classic game Tetris. It is created by using minimum classes and functions. OpenGL has been used effectively for all the animation without the use of any additional libraries to render the game, for animation and to perform different game actions such as change the shape of the block, quit or to reset.

6. Bibliography

Open GL Functions
http://www.idc-online.com/technical_references/pdfs/information_technology/Opengl_control_functions.pdf

OpenGL Colours and Primitives
http://www.dgp.toronto.edu/~ah/csc418/fall_2001/tut/ogl_draw.html

OpenGL Attributes
https://www.cs.utexas.edu/~fussell/courses/cs324e2003/hear50265_ch04.pdf

Appendix A - Source Code

Board.cpp:

```
#include "Board.h"
#include "constants.h"
#include "include/Angel.h"
#include <iostream>
#include <cstring>
using namespace std;
Board::Board(){
    reset();
}
void Board::reset(){
    for (int y = 0; y < 20; ++y) {
        for (int x = 0; x < 10; ++x) {
            blocks[y][x] = kBlockEmpty;
        }
    }
    num_of_points = 0;
}
// current tetromino x position and the shape array check for collision
bool Board::has_collision(bool tetro_blocks[4][4], int steps, int cur_x){
```

```
    int left_most = 999;
    int right_most = -999;
    int bottom_most = -999;
    for (int y = 0; y < 4; ++y) {
        for (int x = 0; x < 4; ++x) {
            if (tetro_blocks[y][x]) {
                if (x < left_most) left_most = x;
                if (x > right_most) right_most = x;
                if (y > bottom_most) bottom_most = y;
            }
        }
    }

    // check side border, left_most and right_most are between 0~3
    if (cur_x + left_most < 0) return true;
    if (cur_x + right_most > 9) return true;

    // check bottom border
    if (steps + bottom_most >= 20)
        return true;

    // check collision wrt existing tetromino
    for (int y = 0; y < 4; ++y) {
        for (int x = 0; x < 4; ++x) {
            if (steps + y >= 0 && steps + y < 20 &&tetro_blocks[y][x]
                && (blocks[steps + y][cur_x + x] != kBlockEmpty)) {
                return true;
            }
        }
    }

    return false;
}
```

```
bool Board::top_reached(bool tetro_blocks[4][4], int steps){
    int top_most = 999;
    for (int y = 0; y < 4; ++y) {
        for (int x = 0; x < 4; ++x) {
            if (tetro_blocks[y][x]) {
                if (y < top_most) top_most = y;
            }
        }
    }
    if (steps + top_most <= 0) {
        cout<<"reached top";
        return true;
    }
    return false;
}
void Board::add_blocks(bool tetro_blocks[4][4], int steps, int cur_x, int color_id){
    for (int y = 0; y < 4; ++y) {
        for (int x = 0; x < 4; ++x) {
            if (tetro_blocks[y][x] && steps + y < 20 && cur_x + x < 10 && steps + y >= 0) {
                if (blocks[steps + y][cur_x + x] == kBlockEmpty) {
                    num_of_points += 4; // place a square of tetromino
                }
                blocks[steps + y][cur_x + x] = color_id;
                assert(0 <= color_id && color_id < kNumOfColors);
                assert(steps + y >= 0);
                assert(steps + y < 20);
                assert(cur_x + x >= 0);
                assert(cur_x + x < 10);
            }
        }
    }
```

```
    // check full row and remove the row
    for (int y = 0; y < 20; ++y) {
        bool full = true;
        for (int x = 0; x < 10; ++x) {
            if (blocks[y][x] == kBlockEmpty) {
                full = false;
                break;
            }
        }
        if (full) {
            for (int i = y; i > 0; --i) {
                for (int x = 0; x < 10; ++x) {
                    blocks[i][x] = blocks[i - 1][x];
                }
            }
            for (int i = 0; i < 10; ++i)
                blocks[0][i] = kBlockEmpty;
            num_of_points -= 4 * 10;
        }
    }
}
void Board::write_buffer(){
    int current = 0;
    for (int i = 0; i < 20; ++i) {
        for (int j = 0; j < 10; ++j) {
            if (blocks[i][j] != kBlockEmpty) {
                vec2 points[4];
                points[0] = vec2(-W + (j    ) * BLOCK_W, H - (i + 1) * BLOCK_H);
                points[1] = vec2(-W + (j + 1) * BLOCK_W, H - (i + 1) * BLOCK_H);
                points[2] = vec2(-W + (j    ) * BLOCK_W, H - i * BLOCK_H);
                points[3] = vec2(-W + (j + 1) * BLOCK_W, H - i * BLOCK_H);
             glBufferSubData(GL_ARRAY_BUFFER, (kBeginBoardPoints + 4 * current) *
sizeof(vec2), sizeof(points), points);
```

```
            vec4 color = kDefaultColors[blocks[i][j]];
            vec4 colors[4] = {color, color, color, color};

            assert(0 <= blocks[i][j] && blocks[i][j] < kNumOfColors);

              glBufferSubData(GL_ARRAY_BUFFER, kColorsOffset + (kBeginBoardPoints +
4 * current) * sizeof(vec4), sizeof(colors), colors);

            current += 1;
          }
       }
    }
}
```

Game.cpp:

```
#include "Game.h"
#include "constants.h"
#include <sys/time.h>
#include <unistd.h>
Game *Game::singleton = NULL;
void Game::run(int argc, char **argv)
{
 singleton = this;
 timeval t;
 gettimeofday(&t, NULL);
 srand((unsigned)(t.tv_sec * 1000 + t.tv_usec));
 tetromino.game = singleton;
 tetromino.board = &board;
 glutInit(&argc, argv);
#ifdef __APPLE__
  glutInitDisplayMode(GLUT_3_2_CORE_PROFILE | GLUT_RGBA | GLUT_DOUBLE);
#else
 glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);
#endif
```

```
 glutInitWindowSize(340, 600);
 glutCreateWindow("Xetris!");
#ifndef __APPLE__
 glewInit();
#endif
 reset();
 init();
 glutDisplayFunc(display);
 glutKeyboardFunc(keyboard);
 glutSpecialFunc(special);
 glutIdleFunc(idle);
 glutMainLoop();
}
void Game::init()
{
 vec2 points[kTotalPoints];
 for (int i = 0; i < kNumOfHLines; ++i) {
 points[i * 2 ] = vec2(-W, -H + BLOCK_H * i);
 points[i * 2 + 1] = vec2( W, -H + BLOCK_H * i);
 }
 for (int i = 0; i < kNumOfVLines; ++i) {
  points[kNumOfHPoints + i * 2 ] = vec2(-W + BLOCK_W * i, -H);
points[kNumOfHPoints + i * 2 + 1] = vec2(-W + BLOCK_W * i, H);   }
 vec4 colors[kTotalPoints];
 for (int i = 0; i < kNumOfHPoints + kNumOfVPoints; ++i) {
 colors[i] = vec4(0.93, 0.93, 0.93, 1.0);
 }
 GLuint vaoID;
 glGenVertexArrays(1, &vaoID);
 glBindVertexArray(vaoID);
 GLuint vboID;
 glGenBuffers(1, &vboID);
 glBindBuffer(GL_ARRAY_BUFFER, vboID);
     glBufferData(GL_ARRAY_BUFFER, sizeof(points) + sizeof(colors), NULL,
```

```
GL_STATIC_DRAW);
 glBufferSubData(GL_ARRAY_BUFFER, 0, kTotalPoints * sizeof(vec2), points);
glBufferSubData(GL_ARRAY_BUFFER, kColorsOffset, sizeof(colors), colors);

 GLuint program = InitShader("vshader.glsl", "fshader.glsl");
 GLuint vPosition = glGetAttribLocation(program, "vPosition");
 glEnableVertexAttribArray(vPosition);
 glVertexAttribPointer(vPosition, 2, GL_FLOAT, GL_FALSE, 0, BUFFER_OFFSET(0));

 GLuint vColor = glGetAttribLocation(program, "vColor");
 glEnableVertexAttribArray(vColor);
 g l Ve r t e x A t t r i b P o i n t e r ( v C o l o r, 4 , G L _ F L O AT, G L _ FA L S E , 0 ,
BUFFER_OFFSET(kColorsOffset));
 glClearColor(1.0, 1.0, 1.0, 1.0);
}
void Game::reset()
{
 tetromino.interval = kDefaultInterval;
 tetromino.reset();
 board.reset();
 is_game_over = false;
}
void Game::display()
{
 if (singleton->is_game_over) {
 glRasterPos2f(-0.5, 0.1);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,    'G');
glRasterPos2f(-0.2, 0.1);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,    'A');
glRasterPos2f(0.1, 0.1);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,    'M');
glRasterPos2f(0.4, 0.1);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,    'E');
glRasterPos2f(-0.5, -0.2);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,    'O');
```

```
glRasterPos2f(-0.2, -0.2);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,     'V');
glRasterPos2f(0.1, -0.2);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,     'E');
glRasterPos2f(0.4, -0.2);
 glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, 'R');   }
 else {
 singleton->tetromino.write_buffer();
 singleton->board.write_buffer();
 glClear(GL_COLOR_BUFFER_BIT);
    glPolygonMode(GL_FRONT_AND_BACK,     GL_LINE);
glEnable(GL_CULL_FACE);
 glDrawArrays(GL_LINES, 0, kNumOfHPoints + kNumOfVPoints);
 glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
                glDrawArrays(GL_TRIANGLE_STRIP,                kBeginTetrominoPoints,
kNumOfTetrominoPoints);
 glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        glDrawArrays(GL_TRIANGLE_STRIP,        kBeginBoardPoints,        singleton-
>board.num_of_points);
 }
 glutSwapBuffers();
}
void Game::keyboard(unsigned char key, int x, int y)
{
 switch (key) {
 case GLUT_KEY_LEFT:
 singleton->tetromino.left();
 break;
 case GLUT_KEY_RIGHT:
 singleton->tetromino.right();
 break;
 case GLUT_KEY_UP:
 singleton->tetromino.rotate();
 break;
 case GLUT_KEY_DOWN:
```

```
 singleton->tetromino.down();
 break;
 case 'w': case 'W':
 singleton->tetromino.up();
 Break;
 case 'r': case 'R':
 singleton->reset();
 break;
 case 'q': case 'Q':
 case kKeyCodeESC: case kKeyCodeESC2:
 exit(EXIT_SUCCESS);
 break;
 }
}
void Game::special(int key, int x, int y)
{
 singleton->keyboard(key, x, y);  }
void Game::idle()
{
 usleep(20);
 glutPostRedisplay();
}

void Game::game_over()
{
 is_game_over = true;
}
```

Tetris.cpp:

```
#include "Game.h"
int main(int argc, char **argv)
{
 Game game;
 game.run(argc, argv);
```

```
 return 0;
}
```

Tetrismino.cpp:

```
#include "Tetromino.h"
#include "Game.h"
#include "constants.h"
#include "include/Angel.h"
#include <sys/time.h>
#include <cstring>

const bool shapes[28][4] =
{
  {0,  0,  0,  0},
{0,   1,   1,   0},
{0,   1,   1,   0},
{0, 0, 0, 0},

  {0,  0,  0,  0},
{1,   1,   1,   1},
{0,   0,   0,   0},
{0, 0, 0, 0},
  {0,  0,  0,  0},
{0,   0,   1,   1},
{0,   1,   1,   0},
{0, 0, 0, 0},
  {0,  0,  0,  0},
{0,   1,   1,   0},
{0,   0,   1,   1},
{0, 0, 0, 0},

  {0,  0,  0,  0},
{0,   1,   1,   1},
{0,   1,   0,   0},
{0, 0, 0, 0},
```

```
  {0,  0,  0,  0},
{0,   1,   1,   1},
{0,   0,   0,   1},
{0, 0, 0, 0},
  {0,  0,  0,  0},
{0,   1,   1,   1},
{0, 0, 1, 0},
 {0, 0, 0, 0},
};
Tetromino::Tetromino()
{
 color_id = rand() % kNumOfColors;
}
 inline double Tetromino::elapsed() const
{
 timeval t;
 gettimeofday(&t, NULL);
 double elapsedTime;
  elapsedTime = (t.tv_sec - start_time.tv_sec) * 1000.0; // sec to ms
elapsedTime += (t.tv_usec - start_time.tv_usec) / 1000.0; // us to ms
return elapsedTime;
}
void Tetromino::reset()
{
 rotation_count = 0;
 cur_x = 3; // shape outer bound is 4x4
 step_extra = -2;
 shape = (Shape)(rand() % NUM_OF_SHAPES);
 for (int y = 0; y < 4; ++y) {
 for (int x = 0; x < 4; ++x) {
 blocks[y][x] = shapes[shape * 4 + y][x];
 }
 }
 color_id = (color_id + 1) % kNumOfColors;
 gettimeofday(&start_time, NULL);
```

```
if (interval > kMinimumInterval) {
 interval /= kIntervalSpeedUp;
 }
}
void Tetromino::left()
{
 if (!board->has_collision(blocks, _steps(), cur_x - 1)) {
 cur_x -= 1;
 }
}
void Tetromino::right()
{
  if (!board->has_collision(blocks, _steps(), cur_x + 1)) {
cur_x += 1;
 }
}
void Tetromino::rotate()
{
 rotation_count += 1;
 if (shape == O) {
 ;}
  else if (shape == I || shape == S || shape == Z) {
if (rotation_count % 2 == 0) {
 _rotate_back();
 }
 else {
 _rotate_ccw();
 }
 }
  else if (shape == L || shape == J || shape == T) {
_rotate_ccw();
 }
}
void Tetromino::_rotate_ccw()
{
```

```
 bool new_blocks[4][4] = {{0}};
 for (int y = 0; y < 4; ++y) {
 for (int x = 0; x < 4; ++x) {
 if (blocks[y][x]) {
 int new_y = 1 - (x - 2);
 int new_x = 2 + (y - 1);
 if (0 <= new_y && new_y < 4 &&
 0 <= new_x && new_x < 4) {
 new_blocks[new_y][new_x] = 1;   }

 assert(new_y >= 0);
 assert(new_y < 4);
 assert(new_x >= 0);
 assert(new_x < 4);
 }
 }
 }
  if (!board->has_collision(new_blocks, _steps(), cur_x)) {
for (int y = 0; y < 4; ++y) {
 for (int x = 0; x < 4; ++x) {
 blocks[y][x] = new_blocks[y][x];
 }
 }
 }
 else {
 rotation_count -= 1;
 }
}
void Tetromino::_rotate_back()
{
 bool new_blocks[4][4] = {{0}};
 for (int y = 0; y < 4; ++y) {
 for (int x = 0; x < 4; ++x) {
   new_blocks[y][x]  =  shapes[shape  *  4  +  y][x];
```

```
assert(shape * 4 + y < 28);
 }
 }
  if (!board->has_collision(new_blocks, _steps(), cur_x)) {
memcpy(blocks, new_blocks, 4 * 4);
 }
 else {
 rotation_count -= 1;
 }
}
void Tetromino::up()
{
 step_extra -= 1;
}
void Tetromino::down()
{
 step_extra += 1;
  if (board->has_collision(blocks, _steps(), cur_x)) {
_add_blocks();
 }
}
void Tetromino::_add_blocks()
{
 int rollback = 1;
  while (board->has_collision(blocks, _steps() - rollback, cur_x)) {
rollback += 1;
 }
  board->add_blocks(blocks, _steps() - rollback, cur_x, color_id);
reset();
}

int Tetromino::_steps()
{
 return ceil(elapsed() / interval + step_extra);
}
```

```
void Tetromino::write_buffer()
{
 int steps = _steps();
 if (board->has_collision(blocks, steps, cur_x)) {   if
(board->top_reached(blocks, steps)) {
 _add_blocks();
 game->game_over();
 return;
 }
 else {
 _add_blocks();
 return;
 }
 }
 int current = 0;
 for (int i = 0; i < 4; ++i) {
 for (int j = 0; j < 4; ++j) {
 if (blocks[i][j]) {
 vec2 points[4];
  points[0] = vec2(-W + (j + cur_x ) * BLOCK_W, H - (i + 1) * BLOCK_H - steps  *
BLOCK_H);
 points[1] = vec2(-W + (j + cur_x + 1) * BLOCK_W, H - (i + 1) * BLOCK_H -  steps *
BLOCK_H);
 points[2] = vec2(-W + (j + cur_x ) * BLOCK_W, H - i * BLOCK_H - steps *  BLOCK_H);
  points[3] = vec2(-W + (j + cur_x + 1) * BLOCK_W, H - i * BLOCK_H - steps *
BLOCK_H);
   glBufferSubData(GL_ARRAY_BUFFER, (kBeginTetrominoPoints + 4 * current) *
sizeof(vec2), sizeof(points), points);
 vec4 color = kDefaultColors[color_id];
 vec4 colors[4] = {color, color, color, color};
 glBufferSubData(GL_ARRAY_BUFFER, kColorsOffset +  (kBeginTetrominoPoints + 4 *
current) * sizeof(vec4), sizeof(colors), colors);   current += 1;
 }
 }
 }}
```