CS321-Final Project

Genetic Algorithm Visualization – Tanner Stevens

Contents

[2 Overview 2](#_Toc437428543)

[3 Visualization 2](#_Toc437428544)

[4 Genetic Algorithm 3](#_Toc437428545)

[5 Command List 3](#_Toc437428546)

[5.1 Modeling 3](#_Toc437428547)

[5.1.1 Tetris Inspector Window 3](#_Toc437428548)

[5.2 Geometric Transformations 3](#_Toc437428549)

[5.3 Mouse 4](#_Toc437428550)

[5.3.1 Main Window 4](#_Toc437428551)

[5.3.2 Tetris Inspector 4](#_Toc437428552)

[5.4 Other Operations 4](#_Toc437428553)

[5.4.1 Main Window 4](#_Toc437428554)

[6 Utilized Graphic Techniques 4](#_Toc437428555)

# Overview

My Final Project is centered on a Genetic Algorithm which attempts to play Tetris and visualizing the data generated by that Genetic Algorithm. The Graphical Output consists of two windows, the first of which the “Main Window” consists of the main visualization component as it displays representations of each move set, as a Pyramid, and allows for displaying multiple simulations at once. The second of the two windows is a “Tetris Inspector” this allows for reviewing move sets generated by the Genetic Algorithm, initially there is no move set bound to the “Tetris Inspector” thus in order to bind a move set to be reviewed the user must click on the base of a move sets pyramid, the graphical representation of the move sets, on the main screen.

# Visualization

Each Tetris Simulation is graphically represented by a Visualizer, and the Main Screen consists of all visualizers. Each Visualizer consists of Pyramids which are the graphical representation of each move set that Simulation is currently evaluating. Each Pyramid has 3 distinct features; Area(Base Size), Height, and Base Border. The values of these features are dependent on the move sets evaluation. Each Pyramid’s Material ‘Koefficients’, Ambient, Diffuse, Specular and Shiny and even Emission, are then based off of the move sets Gene’s, thus allowing for seeing relations between move sets from Color Attributes. The Main Screen also has a pretty neat background that is texture mapped using the SOIL Library, and also has a single static positional light source.

The Tetris Inspector screen then also has a background that is textured mapped via SOIL, but more importantly it allows for the 3D viewing of the Tetris move sets. Once the user has bound a move set to the inspector, by clicking on the base of the move set’s pyramid on the main screen, hitting space while the Tetris Inspector screen is active will advance the move set by one move, the score will then be displayed in the title of the Tetris Inspector. For the sake of creating an interesting effect while fulfilling Project Requirements the Tetris Inspector screen also features a ‘Flash Light’ which has a small cutoff angle for a spotlight and dynamic positioning based on the mouse cursor position. As a technical note the Tetris Board is represented by occupied space and not by each individual piece itself, thus the board/pieces are comprised of glutDodecahedrons currently but the user can swap them to Dodecahedrons loaded from an .obj with such possible the Tetris Board could literally be represented using any .obj model.

\*Both windows feature a Geometric Transformation Suite allowing Visualizers to have Geometric Transformations applied individually or globally, as-well as the Tetris Inspector window applies Geometric Transformations to the board of Tetris Pieces.

# Genetic Algorithm

Although this is not the topic of the class given that it’s the back end of my Final Project I’ll provide a brief description of the Genetic Algorithm.

The Genetic Algorithm attempts to play Tetris by creating move sets from two random number seeds, a position gene and a rotation gene. These genes are contained within a DNA class, and each Tetris Player class manages generations of DNA, with the generation sized currently tuned to 100. Thus the Genetic Algorithm begins by randomly assigning the genes to the first generation of DNA, then the succeeding generations genes are determined by that players highest scoring DNA and high weighted scoring DNA, those DNA’s genes are intermixed with the rest of the generations genes, and further mutated based on how long the player has plateaued on a certain highscore.

# Command List

## Modeling

### Tetris Inspector Window

/ Toggle between .obj and glut Dodecahedrons

## Geometric Transformations

w,s Move Object Upwards/Downwards

a,d Move Object to the Left/Right

q,e Move Object Outwards/Inwards

8 ,2 Positive/Negative Y-axis Rotation

4,6 Positive/Negative X-axis Rotation

7,9 Positive/Negative Z-axis Rotation

+,- Scale Object Up/Down

## Mouse

### Main Window

Left mouse button Bind move set to Tetris Inspector Window

### Tetris Inspector

Mouse Movement Moves the Flash Light

## Other Operations

### Main Window

p Pause Visualizer under Mouse Cursor

\ Remove Visualizer under Mouse Cursor

, Visualizer under mouse Cursor goes into Delayed Drawing Mode

. Visualizer Draws in Real Time (No Delay)

ALT Commands affect all Visualizers

ALT+\* Add a Visualizer

ALT+/ Remove the youngest Visualizer

# Utilized Graphic Techniques

void **glBegin**(GLenum  *cap*);

|  |  |  |
| --- | --- | --- |
| void **glDisable**( | GLenum | *cap*); |

*cap*

Specifies a symbolic constant indicating a GL capability.

void **glClear**(GLbitfield  *mask*);

*mask*

Bitwise OR of masks that indicate the buffers to be cleared. The four masks are GL\_COLOR\_BUFFER\_BIT, GL\_DEPTH\_BUFFER\_BIT, GL\_ACCUM\_BUFFER\_BIT, andGL\_STENCIL\_BUFFER\_BIT.

|  |  |
| --- | --- |
| void **glMatrixMode**( | GLenum *mode*); |

*mode*

Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: GL\_MODELVIEW, GL\_PROJECTION, and GL\_TEXTURE. The initial value is GL\_MODELVIEW. Additionally, if the ARB\_imaging extension is supported, GL\_COLOR is also accepted.

void **glPushMatrix**(*void*);

void **glPopMatrix**(*void*);

void **glViewport**(GLint  *x*, GLint  *y*, GLsizei  *width*, GLsizei  *height*);

*x*, *y*

Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0,0).

*width*, *height*

Specify the width and height of the viewport. When a GL context is first attached to a window, *width* and *height* are set to the dimensions of that window.

void **glVertex3f**(GLfloat *x*, GLfloat *y*, GLfloat *z*);

*x*, *y*, *z*, *w*

Specify *x*, *y*, *z*, and *w* coordinates of a vertex. Not all parameters are present in all forms of the command.

|  |  |
| --- | --- |
| void **glVertex3fv**( | const GLfloat \* *v*); |

*v*

Specifies a pointer to an array of two, three, or four elements. The elements of a two-element array are x and y; of a three-element array, x, y, and z; and of a four-element array, x, y, z, and w.

void **glMateriali**(GLenum *face*, GLenum *pname*, GLint *param*);

*face*

Specifies which face or faces are being updated. Must be one of GL\_FRONT, GL\_BACK, orGL\_FRONT\_AND\_BACK.

*pname*

Specifies the single-valued material parameter of the face or faces that is being updated. Must be GL\_SHININESS.

*param*

Specifies the value that parameter GL\_SHININESS will be set to.

void **glMaterialfv**(GLenum *face*, GLenum *pname*, const GLfloat \**params*);

*face*

Specifies which face or faces are being updated. Must be one of GL\_FRONT, GL\_BACK, orGL\_FRONT\_AND\_BACK.

*pname*

Specifies the material parameter of the face or faces that is being updated. Must be one ofGL\_AMBIENT, GL\_DIFFUSE, GL\_SPECULAR, GL\_EMISSION, GL\_SHININESS, GL\_AMBIENT\_AND\_DIFFUSE, or GL\_COLOR\_INDEXES.

*params*

Specifies a pointer to the value or values that *pname* will be set to.

void **glColor3f**( GLfloat *red*, GLfloat *green*, GLfloat *blue*);

*red*, *green*, *blue*

Specify new red, green, and blue values for the current color.

*alpha*

Specifies a new alpha value for the current color. Included only in the four-argumentglColor4 commands.

void **glFrustum**(GLdouble *left*, GLdouble *right*, GLdouble *bottom*, GLdouble *top*, GLdouble *nearVal*, GLdouble *farVal*);

*left*, *right*

Specify the coordinates for the left and right vertical clipping planes.

*bottom*, *top*

Specify the coordinates for the bottom and top horizontal clipping planes.

*nearVal*, *farVal*

Specify the distances to the near and far depth clipping planes. Both distances must be positive.

void **glGetDoublev**(GLenum *pname*, GLdouble \* *params*);

*pname*

Specifies the parameter value to be returned. The symbolic constants in the list below are accepted.

*params*

Returns the value or values of the specified parameter.

void **glLightf**(GLenum *light*, GLenum *pname*, GLfloat *param*);

*light*

Specifies a light. The number of lights depends on the implementation, but at least eight lights are supported. They are identified by symbolic names of the form GL\_LIGHT i, where i ranges from 0 to the value of GL\_MAX\_LIGHTS - 1.

*pname*

Specifies a single-valued light source parameter for *light*. GL\_SPOT\_EXPONENT,GL\_SPOT\_CUTOFF, GL\_CONSTANT\_ATTENUATION, GL\_LINEAR\_ATTENUATION, andGL\_QUADRATIC\_ATTENUATION are accepted.

*param*

Specifies the value that parameter *pname* of light source *light* will be set to.

void **glPointSize**(GLfloat *size*);

*size*

Specifies the diameter of rasterized points. The initial value is 1.

void **glRotated**(GLdouble *angle*, GLdouble *x*, GLdouble *y*, GLdouble *z*);

*angle*

Specifies the angle of rotation, in degrees.

*x*, *y*, *z*

Specify the *x*, *y*, and *z* coordinates of a vector, respectively.

void **glScaled**(GLdouble *x*, GLdouble *y*, GLdouble *z*);

*x*, *y*, *z*

Specify scale factors along the *x*, *y*, and *z* axes, respectively.

void **glTranslated**(GLdouble *x*, GLdouble *y*, GLdouble *z*);

*x*, *y*, *z*

Specify the *x*, *y*, and *z* coordinates of a translation vector.

void **glTexCoord2f**(GLfloat  *s*, GLfloat  *t*);

*s*, *t*, *r*, *q*

Specify *s*, *t*, *r*, and *q* texture coordinates. Not all parameters are present in all forms of the command.

void **glBindTexture**(GLenum *target*, GLuint *texture*);

*target*

Specifies the target to which the texture is bound. Must be either GL\_TEXTURE\_1D,GL\_TEXTURE\_2D, GL\_TEXTURE\_3D, or GL\_TEXTURE\_CUBE\_MAP.

*texture*

Specifies the name of a texture.

GLint **gluUnProject**(GLdouble *winX*, GLdouble *winY*, GLdouble *winZ*, const GLdouble \* *model*, const GLdouble \* *proj*, const GLint \* *view*, GLdouble\* *objX*, GLdouble\* *objY*, GLdouble\* *objZ*);

*winX*, *winY*, *winZ*

Specify the window coordinates to be mapped.

*model*

Specifies the modelview matrix (as from a [glGetDoublev](https://www.opengl.org/sdk/docs/man2/xhtml/glGetDoublev.xml) call).

*proj*

Specifies the projection matrix (as from a [glGetDoublev](https://www.opengl.org/sdk/docs/man2/xhtml/glGetDoublev.xml) call).

*view*

Specifies the viewport (as from a [glGetIntegerv](https://www.opengl.org/sdk/docs/man2/xhtml/glGetIntegerv.xml) call).

*objX*, *objY*, *objZ*

Returns the computed object coordinates.