Protobyte API

Contributors: Brittni L. Watkins

***API Key:***

* ***Headers***
* **/Directories/and/Paths.ext**
* *Code*
* Menu Options
* Problem/Question

***Variables:***

* Project Title
  + The title of your project
* projectName
  + Name of your project. Will be used to name main project .cpp and .h files

***Beginning a New Project:***

1. Creating Project Files
   1. Open terminal window
   2. Navigate to **project\_wizard** directory of Protobyte
      1. Current Absolute Path
         1. **/Dropbox/ira\_dev/Protobyte\_0.1.0\_quark/Protobyte/project\_wizard/**
   3. Execute command *./wizard projectName*
      1. Creates .cpp and .h files in **/Protobyte\_0.1.0\_quark /Protobyte/src**
2. Adding Project Files to Protobyte Environment
   1. Open Visual Studios
   2. Open project file
      1. Current Absolute Path
         1. **/Dropbox/ira\_dev/Protobyte\_0.1.0\_quark/protobyte\_dev\_WIN/Zhlong\_dev\_WIN/Shlonh\_dev\_WIN.vcxproj**
   3. Add new project files to Visual Studios Project
      1. Right click **srcProtobyte** folder under **Protobyte\_Project**
      2. Hover over Add menu option
      3. Select Existing Item
      4. Navigate to **src** directory
         1. Current Absolute Path
            1. **/Dropbox/ira\_dev/Protobyte\_0.1.0\_quark/Protobyte/src/**
      5. Select and Add projectName.h and projectName.cpp
3. Running new project
   1. Open ProtoMain.cpp
   2. At top of document
      1. *#include “projectName.h”*
   3. In main()
      1. *ijg::ProtoPlasm objectName(canvasWidth, canvasHeight, “Project Title”, new projectName());*
      2. Note: canvasWidth, canvasHeight, and Project Title can only be changed in ProtoMain.cpp

***General Information about Visual Environment:***

* Origin (0, 0) is at the center of the window
* Default background color is black
* Default stoke color is black
* Default fill color is white
* RGB is on a 0-1 scale
* Greyscale
  + 0 = black
  + 1 = white

***Console Output:***

void print(…, …);

* Prints parameters to the console
* Example:
  + *print(“x =”, x);*

void println(…);

* prints parameters to the console followed by a new line character
* Example:
  + *println(“y =”, y);*

void trace(…);

* prints parameters to the console followed by a new line character
* performs the same function as *println(…)*
* Example:
  + *trace(“z =”, z);*

***Visual Environment:***

Variables:

* Note: the changes made to these variables only apply to those shapes drawn after the changes are made.

bool isStroke

* When set to *true*, shows the shapes’ borders in the current *strokeColor*
* When set to *false*, shapes’ borders do not appear

bool isFill

* When set to *true*, shows the shapes’ fill in the current *fillColor*
* When set to false, shapes’ fill does not appear
  + Shape becomes “see-through”

float lineWidth

* Sets the width of the shapes’ borders
* Example
  + *lineWidth = 2.0;*
  + *lineWidth = 1.0;*

Col4f fillColor

* sets the shapes’ fill color
* Example:
  + *fillColor = Col4f(0.2, 0.5, 0.1, 1.0);*

Col4f strokeColor

* sets the shapes’ borders’ color
* Example:
  + *strokeColor = Col4f(1.0, 0.0, 0.0, 1.0);*

Functions:

int getWidth();

* returns the width of the window/canvas
* returns same value as *getCanvasWidth()*

int getHeight();

* returns the height of the window/canvas
* returns same value as *getCanvasHeight()*

Dim2i getSize();

* returns a Dim2i object containing the width and height of the window/canvas
* returns same value as *getCanvasSize()* and *getWindowFrameSize()*

int getCanvasWidth();

* returns the width of the window/canvas
* returns same value as *getWidth()*

int getCanvasHeight();

* returns the height of the window/canvas
* returns same value as *getHeight()*

Dim2i getCanvasSize();

* returns a Dim2i object containing the width and height of the window/canvas
* returns same value as *getSize()* and *getWindowFrameSize()*

Dim2i getWindowFrameSize();

* returns a Dim2i object containing the width and height of the window/canvas
* returns same value as *getSize()* and *getCanvasSize()*

void background(float color);

* sets the background of the window
* performs same function as *setBackground(…)*
* Alternative methods of invocation:
  + background(float r, float g, float b);
  + background(Col3f color);
  + background(Col4f color);

void setBackground(float color);

* sets the background of the window
* performs the same function as *background(…)*
* Alternative methods of invocation:
  + setBackground(float r, float g, float b);
  + setBackground(Col3f color);
  + setBackground(Col4f color);

void stroke(float color);

* Sets *isStroke* equal to *true*
* Sets color of shapes’ borders
* Applies to all shapes drawn after the function is called
* Performs same function as *isStroke = true; strokeColor = …;*
* Alternative methods of invocation:
  + stroke(float r, float g, float b);
  + stroke(float r, float g, float b, float a);
  + stroke(float color, float a);
  + stroke(Col4f color);

void strokeWeight(float lineWidth);

* Sets width of shapes’ borders
* Applies to all shapes drawn after the function is called
* Performs the same function as *lineWidth = …;*

void noStroke();

* Sets *isStroke* equal to *false*
* Performs same function as *isStroke = false;*

void fill(float color);

* Sets *isFill* equal to *true*
* Sets the shapes’ fill color
* Applies to all shapes drawn after the function is called
* Performs same function as *isFill = true; fillColor = …;*
* Alternative methods of invocation:
  + fill(float color, float a);
  + fill(float r, float g, float b);
  + fill(float r, float g, float b, float a);
  + fill(Col4f color);

void noFill();

* sets *isFill* equal to *false*
* Performs same function as *isFill = false;*

void setFrameRate(float frameRate);

float getFrameRate();

* Returns current frame rate of program

int getFrameCount();

* Returns current frame count of program

push();

* Saves current state of the matrix; pushes it onto the stack

pop();

* Resets matrix to its previous state; pops it off of the stack

translate(float tX, float tY, float tZ);

* Assuming that the origin’s current position is at (x, y, z), translates the matrix so that the origin’s position is now at (x + tX, y + tY, z + tZ)
* Alternative methods of invocation:
  + translate(Vec3f tXYZ);

rotate(float angle, float xAxis, float yAxis, float zAxis);

* Assuming that the matrix is currently rotated theta degrees, rotates the matrix to theta + angle degrees
* xAxis, yAxis, and zAxis values
  + range from zero to one
  + express percentage of rotation amongst the three axes
* Rotates matrix around the origin
* Positive angle results in counter-clockwise rotation
* Angles expressed in degrees
* Alternative methods of invocation:
  + rotate(float angle, Vec3f rXYZ);

scale(float sX, float sY, float sZ);

* Assuming that the current matrix is scaled (x, y, z), scales the matrix by (x \* sX, y \* sy, z \* sZ)
* Scales toward or away from the origin
* Alternative methods of invocation:
  + scale(float scale);
  + scale(Vec3f sXYZ);

***Objects:***

ProtoDimension2

ProtoDimension3

ProtoColor3

ProtoColor4

ProtoVector2

ProtoVector3

ProtoVector4

ProtoShape

ProtoGeom3

ProtoGeom2

ProtoEllipse

ProtoBlock

ProtoCylinder

ProtoShere

ProtoToroid

ProtoGeosphere

ProtoCurve3

ProtoPath3

ProtoTube

ProtoGroundPlane

ProtoPlane

ProtoPoint3

***Mouse/Key Events:***

Global Variables:

float mouseX

float mouseY

float mouseLastFrameX

float mouseLastFrameY

int mouseButton

int mouseAction

int mouseMods

bool isMousePressed

Functions:

void keyPressed()

void mousePressed() {}

void mouseRightPressed()

void mouseReleased()

void mouseRightReleased()

void mouseMoved()

void mouseDragged()

void arcBallBegin();

void arcBallEnd();

***Two Dimensional Primitives:***

Rectangle

* Note: Each method has an additional, optional parameter that can alter the registration of the rectangle being drawn
  + The following may be passed into the rect(…) functions in order to change the shape’s registration
    - CENTER
    - CORNER
    - CORNER\_TR
    - CORNER\_BL
    - CORNER\_BR

void rect(float radius1, float radius2);

* Draws a rectangle centered at origin with width of *radius1* and height of *radius2*
* Default registration is *CENTER*
* Alternative methods of invocation:
  + *rect(float radius1, float radius2, Registration reg);*

void rect(Vec2 pt0, Vec2 pt1);

* Draws a rectangle whose top left corner is located at (*pt0.x*, *pt0.y*) and whose bottom right corner is located at (*pt1.x*, *pt1.y*)
* Default registration is *CORNER*
* Alternative methods of invocation
  + *rect(Vec2 pt0, Vec2 pt1, Registration reg);*

void rect(float x, float y, float w, float h);

* Draws a rectangle whose top left corner is located at (*x*, *y*) with a width of *w* and a height of *h*
* Default registration is *CORNER*
* Alternative methods of invocation:
  + *rect(float x, float y, float w, float h, Registration reg);*

Quadrilateral

* Note: The vertices of quadrilaterals should be drawn counter-clockwise or in the following order: top-left, bottom-left, bottom-right, top-right.

void quad(float x0, float y0, float x1, float y1, float x2, float y2, float x3, float y3);

* Draws a quadrilateral with vertices at (*x0*, *y0*), (*x1*, *y1*), (*x2*, *y2*), (*x3*, *y3*)
* Default registration is *CENTER*
* Alternative methods of invocation:
  + *quad(float x0, float y0, float x1, float y1, float x2, float y2, float x3, float y3, Registration reg);*

void quad(Vec2 pt0, Vec2 pt1, Vec2 pt2, Vec2 pt3);

* Draws a quadrilateral with vertices located at (*pt0.x*, *pt0.y*), (*pt1.x*, *pt1.y*), (*pt2.x*, *pt2.y*), and (*pt3.x*, *pt3.y*)
* Default registration is *CENTER*
* Alternative methods of invocation:
  + *quad(Vec2 pt0, Vec2 pt1, Vec2 pt2, Vec2 pt3, Registration reg);*

Ellipse

Variables:

int ellipseDetail

Functions:

* Note: Each method has an additional, optional parameter that can alter the registration of the ellipse being drawn
  + The following may be passed into the ellipse(…) functions in order to change the shape’s registration
    - CENTER
    - CORNER
    - CORNER\_TR
    - CORNER\_BL
    - CORNER\_BR

void ellipse(float x, float y, float w, float h);

* Draws an ellipse centered at (*x*, *y*) with a width of *w* and height of *h*
* Default registration is *CENTER*
* Alternative methods of invocation:
  + *ellipse(float x, float y, float w, float h, Registration reg);*

void ellipse(float x, float y, float r);

* Draws an ellipse centered at (*x*, *y*) with a width and height of *r*
* Default registration is *CENTER*
* Alternative methods of invocation:
  + *ellipse(float x, float y, float r, Registration reg);*

void ellipse(float r);

* Draws an ellipse centered at the origin with a width and height of *r*
* Default registration is *CENTER*
* Alternative methods of invocation:
  + *ellipse(float r, Registration reg);*

void ellipse(float r0, float r1);

* Draws an ellipse centered at the origin with a width of *r0* and a height of *r1*
* Default registration is *CENTER*
* Alternative methods of invocation:
  + *ellipse(float r0, float r1, Registration reg);*

***Suggestions:***

General Information about Visual Environment

* Default background color should be different from default stroke color
  + Maybe have default background color be a shade of grey

Visual Environment

* Overload = operator for *Col4f = Col3f* so that when using global variables to set fill and stroke color, users may pass the variable a *Col4f* object or a *Col3f* object
* Make separate rotate functions for each axis
  + Example: *rotateX(…)*, *rotateY(…)*, *rotateZ(…)*

Two Dimensional Primitives

* Add *line(…)* function
* Add *point(…)* function
* Ellipses
  + Make *ellipseDetail* private
  + Make a *setEllipseDetail(…)* and *getEllipseDetail()* function
* Quadrilaterals
  + Change default registration of *quad(float x0, float y0…)* to *CORNER*
  + Change default registration of *quad(Vec2 pt0, Vec2 pt1…)* to *CORNER*

***Problems:***

Visual Environment

* *setWindowFrameSize(…)* cannot be made private
  + causes error
* *setFrameRate(…)* has no impact on the frame rate of the program

Mouse/Key Events

* *mouseX* and *mouseY* do not reflect that origin is in the center of the window

Two Dimensional Primitives

* Ellipses
  + Ellipse drawing function does not check to make sure that *ellipseDetail* has not changed. Clears and pushes back vector every time an ellipse is drawn. Not efficient
    - Put in check that tests to see if *ellipseDetail* has been changed.