Creation of Sprites – A New Perspective

Sheetal Bandekar
Department of Master of Computer Applications
Gogte Institute of Technology
Udyambag, Belgaum - Karnataka

Vishal Kerkar
Department of Master of Computer Applications
Gogte Institute of Technology
Udyambag, Belgaum - Karnataka

ABSTRACT
In computer graphics the term 'sprite' is a two dimensional picture that is incorporated into a large scene such as a video game. 'Pixel art' or 'Sprite art' deals with creation of sprites. Using the concepts of mathematics especially plane geometry and trigonometry, this research paper gives developers a new logic to create sprites that can be used in a 2D video game to create characters, labels, links, background image, circle and a brick wall. With the proposed latest technology and the new approach towards the creation of sprites a game by name "SCRAPBOOK" has been successfully built and implemented providing a wide scope for future innovation.

Keywords
Sprites, 2D Computer Graphics

1. INTRODUCTION
The term 2D computer graphics involves the creation of digital images from two-dimensional models such as 2D geometric models, text, and digital images and the mechanism used to specify them [3]. Game programming being one of the applications of computer graphics involves first the creation of imaginary objects and then applying transformations to get an animated effect. Often the objects created are imaginary or real-world object which has to be scribbled on a book and then use logic to implement the figment of the programmer's imagination. These objects could be any geometrical shape, text composing of alphabets, or any imaginary characters/ objects. This paper focuses on the creation of the following objects:

1. Characters (alphabets/numbers/special characters).
2. Labels (string of characters).
3. Links to an object.
4. Background which looks like a ruled page consisting of lines.
5. Circle.
6. Brick Wall.

2. PROPOSED TECHNOLOGY
This paper recommends using the technologies like: Python, PyOpenGL, Pyglet, and Stani''s Python Editor a cross-platform integrated development environment (IDE) for the Python programming language. These technologies can be used on Windows OS to implement algorithm to create the specified objects.

2.1 Python
Python is a simple to learn and dynamic programming language with efficient high-level data structures and effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, makes it a perfect language for scripting and rapid application development in many areas and on most platforms. The Python's interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications [4].

2.2 PyOpenGL
OpenGL is an extensively used open and cross-platform library for real-time 3D graphics, which was developed more than twenty years ago. It provides a low-level API that allows the programmers to access the graphics hardware in a uniform way. It is the platform of choice when developing complex 2D or 3D applications that require hardware acceleration that need to work on different platforms. It can be used in a number of languages including C/C++, C#, Java, Objective-C (used in iPhone and iPad games), Python, etc. PyOpenGl is same as OpenGL except that, it is implemented in Python [5].

2.3 Pyglet
Pyglet is a pure python cross-platform application framework intended for game development. It supports windowing, user interface event handling, OpenGL graphics, loading images and videos and playing sounds and music. It works on Windows, OS X and Linux. Pyglet provides an object-oriented programming interface for developing games and other visually-rich applications for Windows, Mac OS X and Linux [6]. Some of the features of Pyglet are:

- No external dependencies or installation requirements. For most application and game requirements, Pyglet needs nothing else besides Python, simplifying distribution and installation [6].
- Take advantage of multiple windows and multi-monitor desktops. Pyglet allows user to use as many windows as one may need, and is fully aware of multi-monitor setups for use with full screen games [6].
- Load images, sound, music and video in almost any format. Pyglet can optionally use AVbin to play back audio formats such as MP3, OGG/Vorbis and WMA, and video formats such as DivX, MPEG-2, H.264, WMV and Xvid [6].

3. LOGIC TO CREATE A CHARACTER
To start with the developer has to decide a basic shape. In this paper a rectangle is used. A rectangle is geometrical shape which has four points with opposite sides equal [2]. Draw a simple rectangle first as shown below.

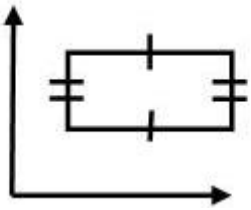

Fig 1: Representation of a simple rectangle

The above rectangle can be represented in detail on a plane with the co-ordinates in terms of its axis as shown below. Here the initial point of the rectangle is (x,y) and size is (xsize,ysize).

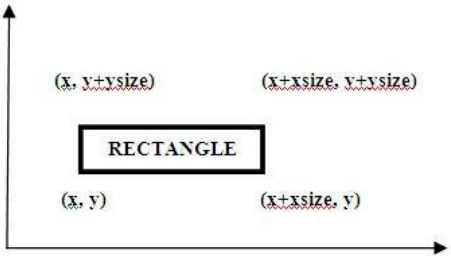

Fig 2: Coordinates of a rectangle

The following snippet is a code to draw a simple rectangle in PyOpenGL. It uses a function gLQuad which is a special type of polygon which accepts 4 points [1].

```
drawRectangle(x,y):
        glBegin()
        gLQuad(x,y)
        gLQuad(x+xsize,y)
        gLQuad(x+xsize,y+ysize)
        gLQuad(x,y+ysize)
        gLEnd()
```

Once the concept of rectangle is clear, arrange the rectangles in a logical way such that it forms the required character. For instance if a programmer wants to draw a character i.e. alphabet 'A', the arrangement of the rectangle could look as shown below.

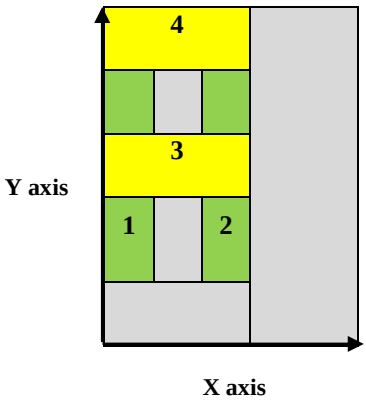

Fig 3: Arrangement of rectangles to form character 'A'

Table 1. Position and Size of Rectangles

Rectangle	Position	Size
1	(0,.1)	(.2,.9)
2	(.4,1)	(.2,.9)
3	(0,.6)	(.6,.1)
4	(0,.9)	(.6,.1)

Now set the axis for the character as shown below:

1px = 5 unit

1 unit = 1 cm

1 px = 5 cm

1/5 = 0.2

1 cm = 1 unit = 0.2 px

X = 1 px = 5 unit

1 unit = 1 cm = 0.2 px

Y = 1 px = 5 unit

1 unit = 1 cm = 0.2 px

Once a character is drawn, store each character in different batches. A Batch is a special type of a file which resides on a hard disk and gets destroyed when the program is terminated. Once batches are created, map it with a dictionary. Dictionaries are similar to what their name suggests - a dictionary. In a dictionary, you have an 'index' of words and for each of them a definition. In Python, the word is called a 'key', and the definition a 'value' [4]. The following line of code shows how to create a batch and map it to the dictionary.

CHAR['A'] = BATCHES_OF_A_CHARACTER

3.1 Algorithm to Create Characters

$$\sum_{i=0}^{n} \text{drawRectangle}(xi,yi)$$

size = (xsize, ysize)

n = no of rectangles

i = 0 (initial rectangles)

(xi, yi) is an initial point of rectangle

Detailed mathematical representation of the above algorithm looks as follows:

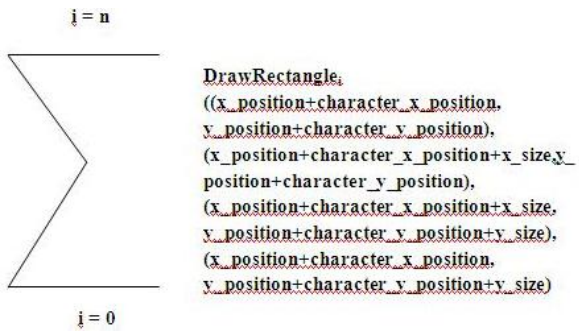

Fig 4: Mathematical representation of the algorithm to create a character

Thus a programmer can use the same technique to create any alphabets, special characters or numbers.

4. LOGIC TO CREATE A LABEL

The term label here means either title of a page, any information which is described using text or it could be a link name that could be used in a game application. A label can be formed by combining the characters (alphabets). Creation of alphabets has been already explained in the previous topics. To create a label first arrange the created characters and give the required spacing between them. To call a label we invoke the function Label() and pass a string as the argument [1].

To display the string, first take the length of a string and keep on moving the character. For example if the programmer wants to draw a label "GIT", then it needs to be first draw on a piece of paper and then find the positions of the characters, size of each character and character spacing among the characters as shown below.

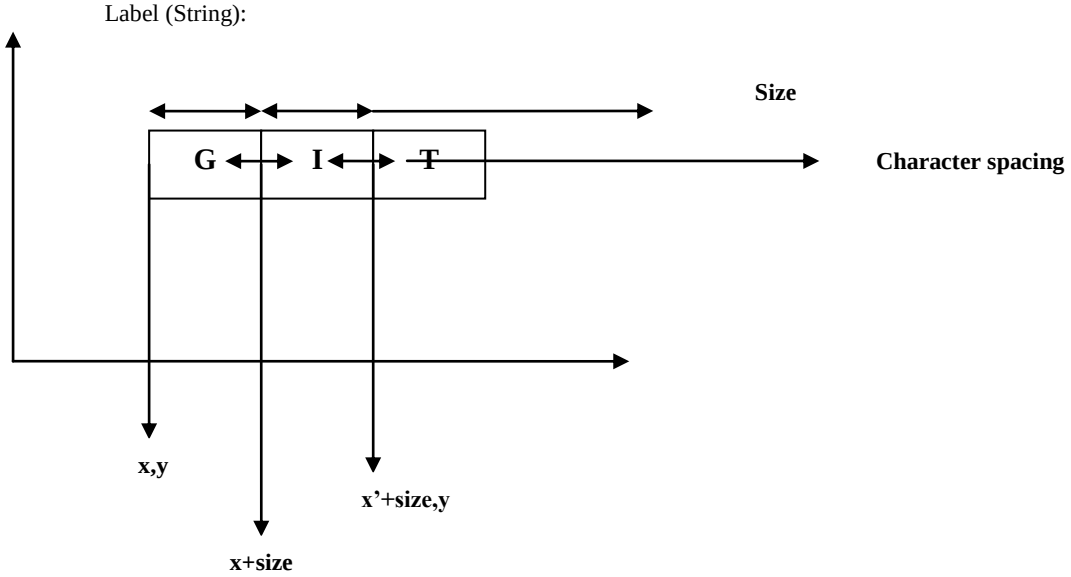

Fig 5: Position of characters "G", "I" and "T"

4.1 Algorithm to Create Characters

Label('G I T'):

x, y = Initial position of x and y

sizex, sizey = Size of character

n = String length

i=0 (initial character)

x_spacing, y_spacing=Character spacing

$$\sum_{i=0}^{n} \textbf{set position x+= i*sizex + x\_spacing}$$

5. LOGIC TO CREATE LINKS

Links here means an event which is invoked after pressing the mouse left button in a specific area. Assume that a user wants to invoke event when left mouse button is pressed within an area of a button as shown below.

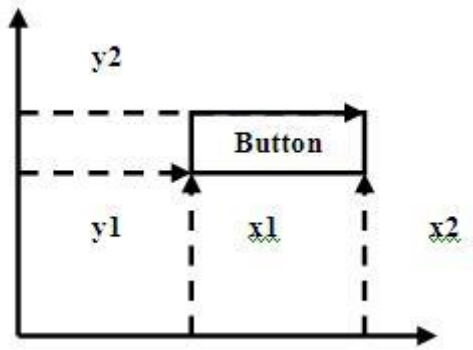

Fig 6: Area of a Button

mouseClick = (mx, my)

The above statement indicates that if "mx" is the area under the button in x-axis and "my" is the area under the button in y-axis, then the event occurs, which implies the following:

if [x1<=mx<=x2 and y1<=my<=y2]

6. LOGIC TO CREATE BACKGROUND

This section proposes to give the logic to create a background which looks like a ruled page which has horizontal and vertical lines. The background here is a layer which is behind the foreground. The foreground in turn has labels on which links are applied as shown below.

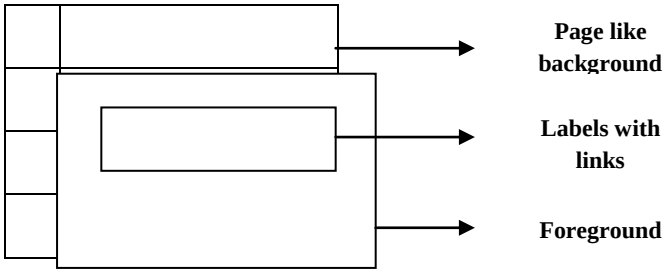

Fig 7: Layers of background and foreground

Since the background is going to look like a ruled 'page like background', there is a necessity to distinguish two types of lines namely: text line and margin line. As one is aware about the line concept, that it has two points, one is a starting point and other is an end point, as shown below.

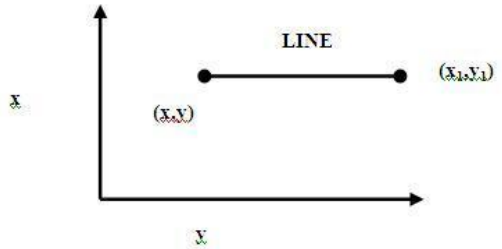

Fig 8: A line on a plane

6.1 Algorithm for Page like Background

n= number of lines (horizontal lines)

i=0 (initial lines)

yindent=indent size for y

since x starts at 0 and ends at x_screen size

x=0, x1=x_screensize

y is incremented with yindent

initially y=y1=yindent

$$\sum_{i=0}^{n}$$ drawline()

6.1.1 Snippet to draw a line

drawline():

```
x=0

x1=x_screensize

y=y1=y_indent

y=y1+=i*y_indent

glBegin()

gLLine(x,y)

gLLine(x1,y1)

gLEnd()
```

6.1.2 Snippet to draw top margin

```
glBegin()

gLLine(x, y+spacing_y)

gLLine(x1, y1+spacing_y)

gLEnd()
```

6.1.3 Snippet to draw left margin

```
x=x1=x_indent

y=0

y1=y_screensize

glBegin()

gLLine(x,y)

gLLine(x1,y1)

# # second line

gLLine(x+x_spacing, y)

gLLine(x1+x_spacing, y1)

gLEnd()
```

7. LOGIC TO CREATE A CIRCLE

A circle can be drawn with n number of points with distance r from the origin (O) with an degree of θ where

0<=θ<=360 as shown below:

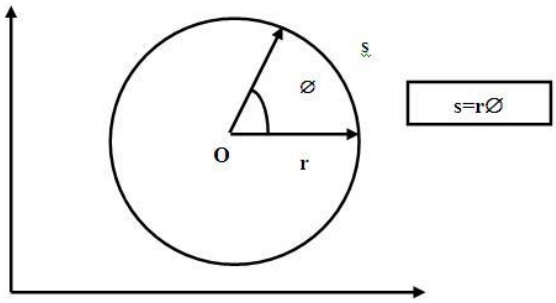

Fig 9: A circle on a plane

Now consider the figure below to be a perfect circle with origin O and radius r_x. Consider a point '**P**' and '**P'**', on the circle with a line joining '**P**' and' **P'**', with origin '**O**'. The line **OP** and **OP'** has an angle θ with the radius r_x. Draw a right angle triangle as shown below and compute the trigonometric values for sine and cosine. Draw a line perpendicular from point P and P' which touches the radius r_x. According to trigonometry the sine(θ) and cosine(θ) are computed as follows:

sin(θ) = Opposite / Hypotenuse

cos(θ) = Adjacent / Hypotenuse

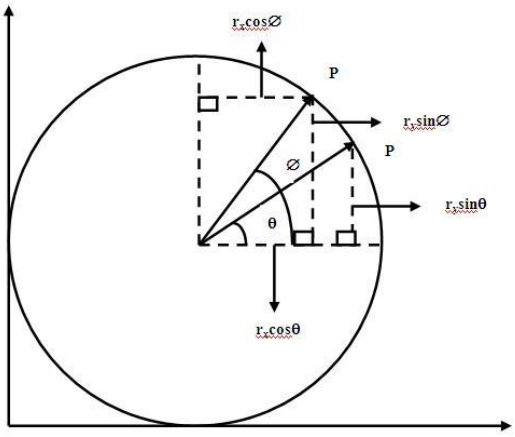

Fig 10: A perfect circle on a plane

7.1 Algorithm to Draw a Circle

Initially θ = 0

Which implies that i=0 and ends at 360

$$\sum_{i=0}^{n=360} \text{gLpolygon}(r_x\cos(i), r_y\sin(i))$$

8. LOGIC TO CREATE A BRICK WALL

To create a Brick Wall, first select the area and position for the wall. It is important to note that a brick is also a rectangle and already the logic to create a rectangle has been described. Resultant brick wall would look as shown below.

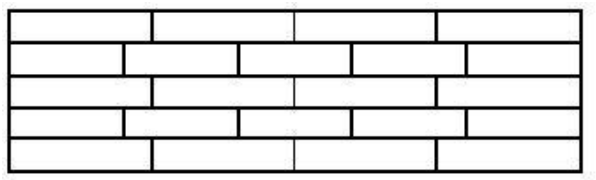

Fig 11: A simple brick wall

The figure below shows the positioning of the wall on a plane.

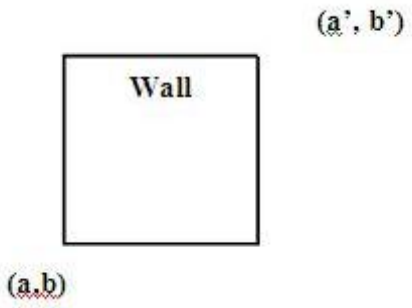

Fig 12: Positioning of a wall on a plane

Now decide the area of brick. For instance, take b_x in x direction and b_y in y direction as shown below:

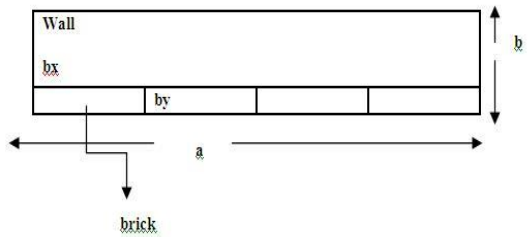

Fig 13: Area of a brick

For instance assume the values for wall and brick's height and weight are as shown below:

Wall height = 10 px

Wall width = 15 px

Brick height = 1 px

Brick width = 2 px

The resultant wall would look as shown below:

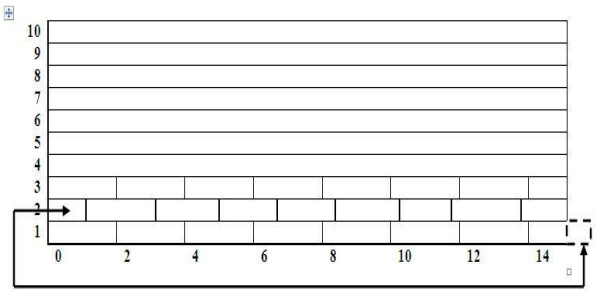

Fig 14: Detailed positioning of bricks

Note the figure above carefully. There is dotted square which is part of the last brick at the lowest portion of the wall. This part of the brick which lies outside the screen of the wall is carry forwarded to upper line of wall (next layer of wall) and then again the regular size bricks are laid besides each other. This process continues till it touches the height of the wall.

8.1 Algorithm to Create a Brick Wall

wallposition = (x,y)

wallsize = (w_x,w_y)

bricksize=(b_x,b_y)

h=y /height of initial wall

temp=x

while($(h+b_y)<(y+w_y)$)

{

w=temp

drawLine(x,$(h+b_y)$,w_x+x,$(h+b_y)$)

while($(w+b_x)<(w_x+x)$)

{

w+=b_x

drawLine(w,h,w,$(h+b_y)$)

}

h+=b_y

temp=x+w_x-w

}

drawLine(x,y+w_y, x+w_x, y+w_y)

while($(w+b_x)<(w_x+x)$)

{

w+=b_x

drawLine(w,h,w,w_y)

}

9. IMPLEMENTATION

The above mentioned algorithms to create characters, labels, links, lines, circle and a brick wall has been implemented in a 2d game by name "SCRAPBOOK". It is a simulator where an air resistance bubble has to pass through the vertical passage without colliding with other objects. Virtual environments have been created with external forces [2]. These forces resist the user with speed. The players are the characters in the environment interacting with virtual environment. In a sense, user is playing against the programmer. The goal of player is to complete the passage in a less period of time without colliding with other objects. The user interface of the welcome page of the game looks as shown below. One can notice ruled page like background, alphabets, and a robot like character. All these have been implemented with the logic provided in this paper.

Fig 15: Welcome page of SCRAPBOOK

The labels "START GAME", "NEW GAME" and "CLOSE" have links to other pages. The label "START GAME" opens a new page which takes user to the main page whose interface looks as shown below.

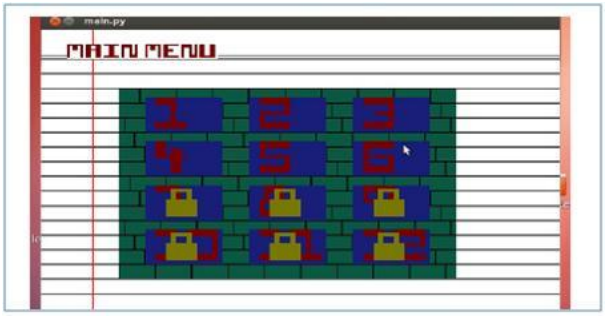

Fig 16: Main Menu page of SCRAPBOOK

The user interface for the "level 1" of the game looks as shown below. Note that the circle (ball) appears in this page which has been built using the algorithm for drawing a circle.

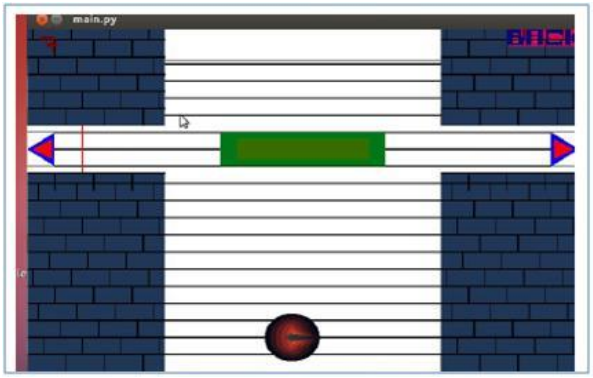

Fig 17: Level 1 of SCRAPBOOK

9.1 Scene Play

While implementing "SCRAPBOOK" a new technique has been used to display the scenes. Whatever one sees in the window is actually a scene. A scene is actually a group of scenes. Every stage is built with object such as a background, a ball, a rocket etc. These objects have respective classes built for it. Every class has its own properties to deal with the environment. For example: a ball will move or burst, a rocket will destroy the target. These classes are deployed when one starts the application, and objects are created for each scene. These objects are later destroyed when user moves to other scene, For example: When a user moves to Menu page the Welcome page object will be destroyed and so forth. Thus this technique basically helps in saving time.

10. CONCLUSION

Every 2D game programmer would prefer to use characters like alphabets, numbers, and special characters from a pool of existing fonts available and also a developer might prefer to use various predefined graphic elements having various formats like .jpg, .png, .gif etc. to build the interface for a game. The proposal in this research paper is a new unique method to create any character, and then group these characters to form labels. Once labels are created the paper describes how to apply links to them. This paper also discusses a new technique to create a ruled 'page like background', creation of a circle and finally how to create a brick wall using the concepts of geometry and trigonometry. The entire proposed algorithm has been implemented to build a 2D game "SCRAPBOOK" and the important interfaces of the game have been displayed.

11. FURTURE SCOPE

This research paper has shown a new perspective towards the creation of a sprite by first explaining how to create a rectangle, and then have shown how the rectangles can be arranged to give a shape of the required sprite. There is a plenty of future scope in this area where a developer can try to research on implementing some other shape like a triangle or any other shape to build a sprite of his/her choice.

12. REFERENCES

[1] Graham Sellers, Richard S. Wright, Jr., and Nicholas Haemel, "OpenGL SuperBible", Addison Wisley; 6 edition (21 July 2013).

[2] David M. Bourg, "Physics for Game Developers", O'Reilly & Associates Inc, 2002.

[3] 2D computer graphics – Wikipedia: http://en.wikipedia.org/wiki/2D_computer_graphics

[4] The Python Tutorial — Python 2.7.8 documentation: https://docs.python.org/2/tutorial/

[5] Cyrille Rossant, "2D graphics rendering tutorial with PyOpenGL" Available: http://cyrille.rossant.net/2d-graphics-rendering-tutorial-with-pyopengl/

[6] pyglet: A cross-platform windowing and multimedia library for Python. Available: http://pyglet.org/