SOL
Shape Oriented Language

Aditya Narayanamoorthi - Language Guru
Gergana Alteva - Project Manager
Erik Dyer - System Architect
Kunal Baweja - Testing
Why SOL?

We wanted:

- a simple, lightweight object-oriented language for creating 2D animations
- the ability to define and create shapes (similar to a class)
- shapes to move as specified by the programmer
- to take away learning a complicated third-party animation tool, such as OpenGL
Advantages to SOL

- Easy to learn
  - similar to Java, C++
- Great alternative to C graphics libraries
  - Skip learning a complex language library
  - Object-oriented
- Easy memory management
  - Programmer does *not* have to worry about memory management
  - No memory leaks
- Abstracts cumbersome features in libraries
  - No renderers, screens, or external media needed to create and animate shapes
Architecture
Stationary Triangle in SDL

```c
// Using SDL, SDL_image, standard IO, math, and strings
#include <SDL.h>
#include <SDL_image.h>
#include <stdio.h>
#include <string>
#include <cmath>

// Screen dimension constants
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;

// Starts up SDL and creates window
bool init();

// Loads media
bool loadMedia();

// Frees media and shuts down SDL
void close();

// Loads individual image as texture
SDL_Texture* loadTexture( std::string path );

// The window we'll be rendering to
SDL_Window* gWindow = NULL;

// The window renderer
SDL_Renderer* gRenderer = NULL;

bool init()
{
    // Initialization flag
    bool success = true;

    // Initialize SDL
    if( SDL_Init( SDL_INIT_VIDEO ) < 0 )
    {
        printf( "SDL could not initialize! SDL Error: %s\n", SDL_GetError() );
        success = false;
    }
    else
    {
        // Set texture filtering to linear
        if( !SDL_SetHint( SDL_HINT_RENDER_SCALE_QUALITY, "1" ) )
        {
            printf( "Warning: Linear texture filtering not enabled!\n" );
        }

        // Create window
        gWindow = SDL_CreateWindow( "SDL Tutorial",
```

```c
```
Stationary Triangle in SDL

```c
int main(int argc, char* argv[])
{
    // Start up SDL and create window
    if (!init())
    {
        printf("Failed to initialize!\n");
    }
    else
    {
        // Load media
        if (!loadMedia())
        {
            printf("Failed to load media!\n");
        }
        else
        {
            // Main loop flag
            bool quit = false;
            // Event handler
            SDL_Event e;
            // While application is running
            while (!quit)
            {
                // Handle events on queue
                while(SDL_PollEvent(&e))
                {
                    // User requests quit
                    if (e.type == SDL_QUIT)
                    {
                        quit = true;
                    }
                }
                // Render green outlined quad
                SDL_FillRect(0,0,SCREEN_WIDTH,SCREEN_HEIGHT);
                SDL_SetRenderDrawColor(gRenderer, 0x00, 0xFF, 0x00, 0x00);
                SDL_RenderDrawRect(gRenderer, gOutlineRect);
                // Update screen
                SDL_RenderPresent(gRenderer);
            }
            // Free resources and close SDL
            close();
            return 0;
        }
    }
}
```
Moving Triangle in SOL

```c
/* Author: Erik Eby */
/* Test Triangle Translate */

    m[0] = (x[0] + y[0]) / 2;
    m[1] = (x[1] + y[1]) / 2;
}

shape Triangle {
    int[2] a;
    int[2] b;
    int[2] c;
    int[2] abm;
    int[2] bcm;
    int[2] acm;

        a = a_init;
        b = b_init;
        c = c_init;
        findCenter(abm, a, b);
        findCenter(bcm, b, c);
        findCenter(acm, a, c);
    }

draw() {
    /* Draw lines between the three vertices of the triangle*/
    drawCurve(a, abm, b, 2, [150, 150, 0]);
    drawCurve(b, bcm, c, 2, [0, 150, 100]);
    drawCurve(c, acm, a, 2, [100, 0, 150]);
}

func main(){
    Triangle t;
    t = shape Triangle([170, 340], [470, 340], [240, 140]);
    t.render = {
        translate([150, 150], 2);
        translate([-130, -130], 3);
        translate([-100, -200], 2);
    }
}
```
Building a Shape

shape Line {
    int[2] a;
    int[2] b;
    int[2] c;

        a = a_init;
        b = b_init;
        c[0] = (a[0] + b[0]) / 2;
        c[1] = (a[1] + b[1]) / 2;
    }

    draw() {
        drawCurve(a, c, b, 2, [0, 0, 0]);
    }
}

→ coordinates represented by int[2]
→ colors by int[3]
→ constructor used to set coordinates
→ define how coordinates will be connected with:
    - drawPoint(int[2], int[3])
    - drawCurve(int[2], int[2], int[2], int, int[3])
    - print(int[2], string, int[3])
→ drawCurve is a bezier curve that accepts 3 control points
func main()
    int[2] dis;
    Line l;
    dis = [200, 0];
    l = shape Line([1,3], [5,8]);

    l.render = {
        translate(dis, 2);
    }
}
DEMO