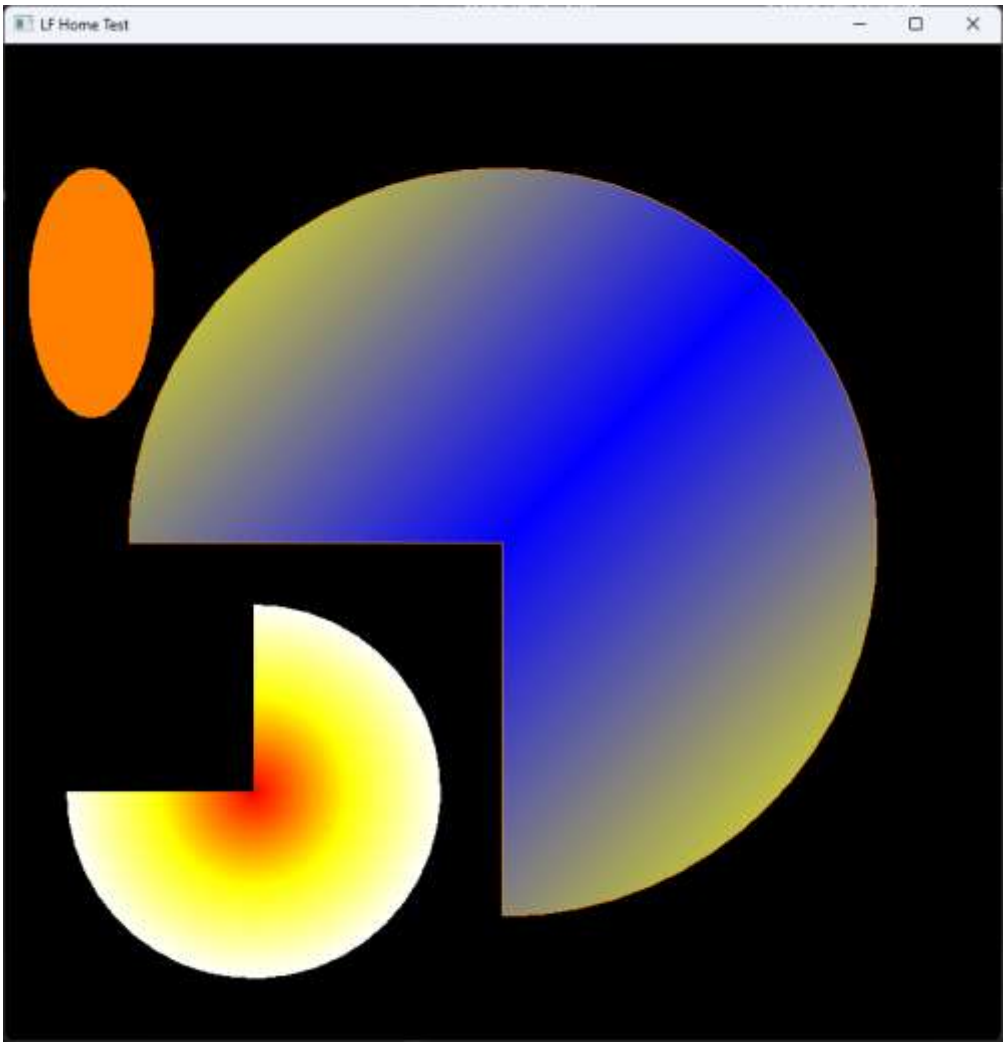
Senior Graphics Engineer Home Test

Sergii Liebodkin



Abstract

The project implements a test task. During the development process, an API for drawing 2D figures was implemented. The key objects are Canvas - the surface for drawing, Path2D - the contour of the figure, as well as brushes for drawing and filling the contours. The contour is defined using such primitives as line, bezier curve and arcade. Introduced brushes for solid, radial and linear fills.

Original task

https://lottiefiles.notion.site/Senior-Graphics-Engineer-Home-Test-e55a3014e3db46c2a9d4ad64e93d06eb

Compilation

VisualStudio 2022

Open solution in *./build* folder

MinGW

Run following command from the root of project folder

```
g++ -I ./deps -I ./deps/khronos ./src/Main.cpp ./src/LottieFiles/Brushes.cpp
./src/LottieFiles/Context.cpp ./src/LottieFiles/OpenGL4.cpp
./src/LottieFiles/Path.cpp ./src/LottieFiles/ShaderSources.cpp -o test -L
./libs/GLFW -mconsole -lopengl32 -lglfw3 -lgdi32
```

API classes

LottieFiles::Context2D - drawing surface

void strokePath(Path2D path, Brush* brush);*

stroke the path by defined brush

void fillPath(Path2D path, Brush* brush);*

fill the path by defined brush

LottieFiles::BrushSolid - solid fill brush

void setColor(unsigned char r, unsigned char g, unsigned char b, unsigned char a);

r - red color component
g - green color component
b - blue color component
a - alpha color component

LottieFiles::BrushRadial - radial gradient using the size, coordinates and center, middle and outer color

void setCenter(float x, float y);

x, y - center of circle

void setRadius(float radius);

radius - radius of circle

void setColorCenter(unsigned char r, unsigned char g, unsigned char b, unsigned char a);

r - red inner color component
g - green inner color component
b - blue inner color component
a - alpha inner color component

void setColorMiddle(unsigned char r, unsigned char g, unsigned char b, unsigned char a);

r - red middle color component
g - green middle color component
b - blue middle color component
a - alpha middle color component

void setColorOuter(unsigned char r, unsigned char g, unsigned char b, unsigned char a);

r - red outer color component
g - green outer color component
b - blue outer color component
a - alpha outer color component

LottieFiles::BrushLinear - liner gradient along the line connecting two given coordinates

void setLinePoints(float p1x, float p1y, float p2x, float p2y);

p1x - x-axis coordinate of the start point.
p1y - y-axis coordinate of the start point.
p2x - x-axis coordinate of the end point.
p2y - y-axis coordinate of the end point.

void setDistance(float distance);

distance - interpolation distance

void setColorInner(unsigned char r, unsigned char g, unsigned char b, unsigned char a);

r - red color component
g - green color component
b - blue color component
a - alpha color component

void setColorOuter(unsigned char r, unsigned char g, unsigned char b, unsigned char a);

r - red color component
g - green color component
b - blue color component
a - alpha color component

LottieFiles::Path2D - list of points for stroke and fill

void beginPath()

reset path points list

void closePath();

close path by adding first point to end of points list

void moveTo(float x, float y);

add point without drawing the line

void lineTo(float x, float y);

add point with drawing a line

void bezierCurveTo(
float cp1x, float cp1y,
float cp2x, float cp2y,
float x, float y,
unsigned int segs = 16);

adds a cubic Bezier curve to the current path. It requires three points: the first two are control points and the third one is the end point.

void arcTo(float x, float y, float radiusX, float radiusY, float startAngle, float endAngle, size_t numSegs = 32);

adds an ellipsed arc to the current path, using the given center point, radiuses and angles

Declaring main entities

```
LottieFiles::Context2D* ctx = nullptr;
LottieFiles::BrushSolid* brushSolid = nullptr;
LottieFiles::BrushRadial* brushRadial = nullptr;
LottieFiles::BrushLinear* brushLinear = nullptr;
LottieFiles::Path2D* pathBezierPie = nullptr;
LottieFiles::Path2D* pathRadialPie = nullptr;
LottieFiles::Path2D* pathEllipse = nullptr;
```

Initializing context and drawing objects

```
// init
void init() {
    // create context
    ctx = new LottieFiles::Context2D();

    // create radial brush
    brushRadial = new LottieFiles::BrushRadial();
    brushRadial->setCenter(200.0f, 200.0f);
    brushRadial->setRadius(150.0f);
    brushRadial->setColorCenter(255, 0, 0, 255);
    brushRadial->setColorMiddle(255, 255, 0, 255);
    brushRadial->setColorOuter(255, 255, 255, 255);

    // create solid brush
    brushSolid = new LottieFiles::BrushSolid();
    brushSolid->setColor(255, 0, 0, 255);

    // create linear brush
    brushLinear = new LottieFiles::BrushLinear();
    brushLinear->setDistance(400.0f);
    brushLinear->setLinePoints(0.0f, 0.0f, 800.0f, 800.0f);
    brushLinear->setColorInner(0, 0, 255, 255);
    brushLinear->setColorOuter(255, 255, 0, 255);

    // create bezier pie path
    pathBezierPie = new LottieFiles::Path2D();
    pathBezierPie->beginPath();
    pathBezierPie->moveTo(400, 100);
    pathBezierPie->bezierCurveTo(566, 100, 700, 234, 700, 400);
    pathBezierPie->bezierCurveTo(700, 566, 566, 700, 400, 700);
    pathBezierPie->bezierCurveTo(234, 700, 100, 566, 100, 400);
    pathBezierPie->lineTo(400, 400);
    pathBezierPie->lineTo(400, 100);

    // create radial pie path
    pathRadialPie = new LottieFiles::Path2D();
    pathRadialPie->beginPath();
    pathRadialPie->moveTo(200, 200);
    pathRadialPie->arcTo(200, 200, 150, 150, -180, 90);
    pathRadialPie->lineTo(200, 200);
    pathRadialPie->closePath();

    // create ellipse path
    pathEllipse = new LottieFiles::Path2D();
    pathEllipse->beginPath();
    pathEllipse->arcTo(70, 600, 50, 100, 0, 360);
}
```

Rendering

```
    // set viewport for context
    ctx->setViewport(width, height);

    // draw radial pie
    brushRadial->setCenter(200.0f, 200.0f);
    brushRadial->setRadius(150.0f);
    brushRadial->setColorCenter(255, 0, 0, 255);
    brushRadial->setColorMiddle(255, 255, 0, 255);
    brushRadial->setColorOuter(255, 255, 255, 255);
    brushSolid->setColor(255, 128, 0, 255);
    ctx->fillPath(pathRadialPie, brushRadial);

    // draw bezier pie
    brushLinear->setDistance(400.0f);
    brushLinear->setLinePoints(0.0f, 0.0f, 800.0f, 800.0f);
    brushLinear->setColorInner(0, 0, 255, 255);
    brushLinear->setColorOuter(255, 255, 0, 255);
    ctx->fillPath(pathBezierPie, brushLinear);
    ctx->strokePath(pathBezierPie, brushSolid);

    // draw ellipse pie
    ctx->fillPath(pathEllipse, brushSolid);
    ctx->strokePath(pathEllipse, brushSolid);
```

Solid brush shader (vertex and fragment)

```
#version 410 core

// attributes
layout (location = 0) in vec2 aPosition;

// uniforms
uniform mat4 uProjMat;

// main
void main()
{
    gl_Position = uProjMat * vec4(aPosition.xy, 0.0, 1.0);
}
```

```
#version 410 core

// uniforms
uniform vec4 uColor;

// outputs
layout (location = 0) out vec4 fragColor;

void main()
{
    fragColor = uColor;
}
```

Radial brush shader (vertex and fragment)

```
#version 410 core

// attributes
layout (location = 0) in vec2 aPosition;

// uniforms
uniform mat4 uProjMat;

// outputs
out vec2 vScreenCoord;

// main
void main()
{
    vScreenCoord = aPosition;
    gl_Position = uProjMat * vec4(aPosition, 0.0, 1.0);
}
```

```
#version 410 core

// inputs
in vec2 vScreenCoord;
// uniforms
uniform vec2 uCenter;
uniform float uRadius;
uniform vec4 uColorCenter;
uniform vec4 uColorMiddle;
uniform vec4 uColorOuter;
// outputs
layout (location = 0) out vec4 fragColor;

void main()
{
    // get distances
    float dist = distance(vScreenCoord, uCenter);
    float halfRadius = uRadius / 2;

    // get color
    vec4 color = uColorOuter;
    if (dist < halfRadius)
        color = mix(uColorCenter, uColorMiddle, dist / halfRadius);
    else if (dist < uRadius)
        color = mix(uColorMiddle, uColorOuter, (dist - halfRadius) / halfRadius);

    // write result
    fragColor = color;
}
```

Linear brush shader (vertex and fragment)

```
#version 410 core

// attributes
layout (location = 0) in vec2 aPosition;

// uniforms
uniform mat4 uProjMat;

// outputs
out vec2 vScreenCoord;

// main
void main()
{
    vScreenCoord = aPosition;
    gl_Position = uProjMat * vec4(aPosition, 0.0, 1.0);
}
```

```
#version 410 core

// inputs
in vec2 vScreenCoord;

// uniforms
uniform vec4 uLinePoints;
uniform float uDistance;
uniform vec4 uColorInner;
uniform vec4 uColorOuter;

// outputs
layout (location = 0) out vec4 fragColor;

void main()
{
    // get distances
    vec2 p0 = vScreenCoord.xy;
    vec2 p1 = uLinePoints.xy;
    vec2 p2 = uLinePoints.zw;

    // https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line
    float dist =
        abs((p2.x - p1.x)*(p1.y - p0.y) - (p1.x - p0.x)*(p2.y - p1.y)) /
        sqrt((p2.x - p1.x)*(p2.x - p1.x) + (p2.y - p1.y)*(p2.y - p1.y));

    // write result
    fragColor = mix(uColorInner, uColorOuter, clamp(dist / uDistance, 0.0,
1.0));
}
```