SKRIPSI “PEMBANGUNAN LAPISAN ABSTRAKSI PADA OPENGL FOR EMBEDDED SYSTEMS SEBAGAI UPAYA MEREDUKSI KOMPLEKSITAS KODE”

1. Abstrak

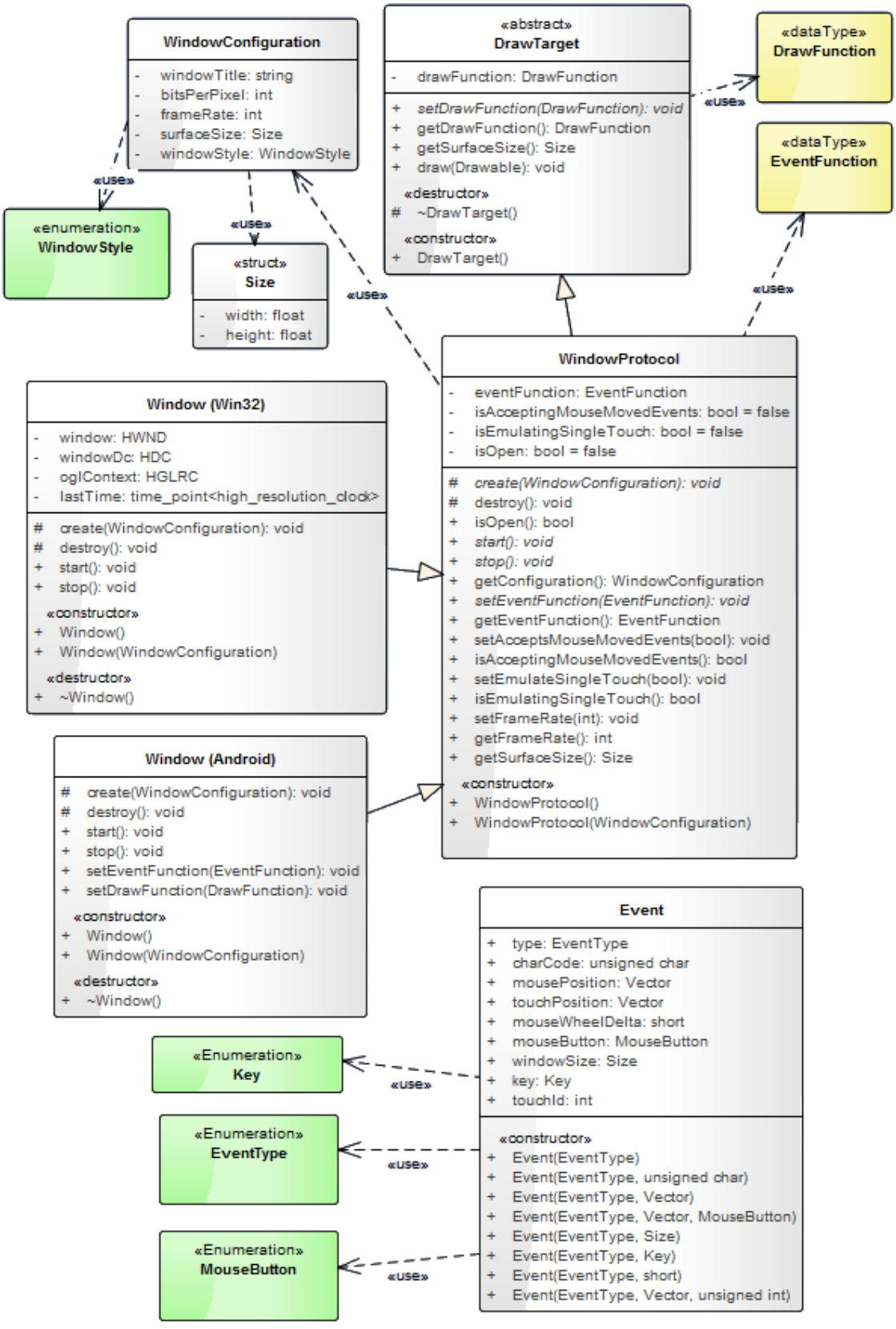
Perkembangan teknologi grafik beberapa tahun terakhir terjadi cukup cepat baik di ranah *desktop* maupun *mobile*. Konten yang dahulu hanya lazim berjalan pada komputer desktop sudah dapat dinikmati pada perangkat *mobile*. Hal tersebut menarik perhatian pengembang aplikasi grafis desktop untuk melebarkan jangkauan aplikasinya ke berbagai platform tanpa membuang waktu untuk melakukan penyesuaian kode.

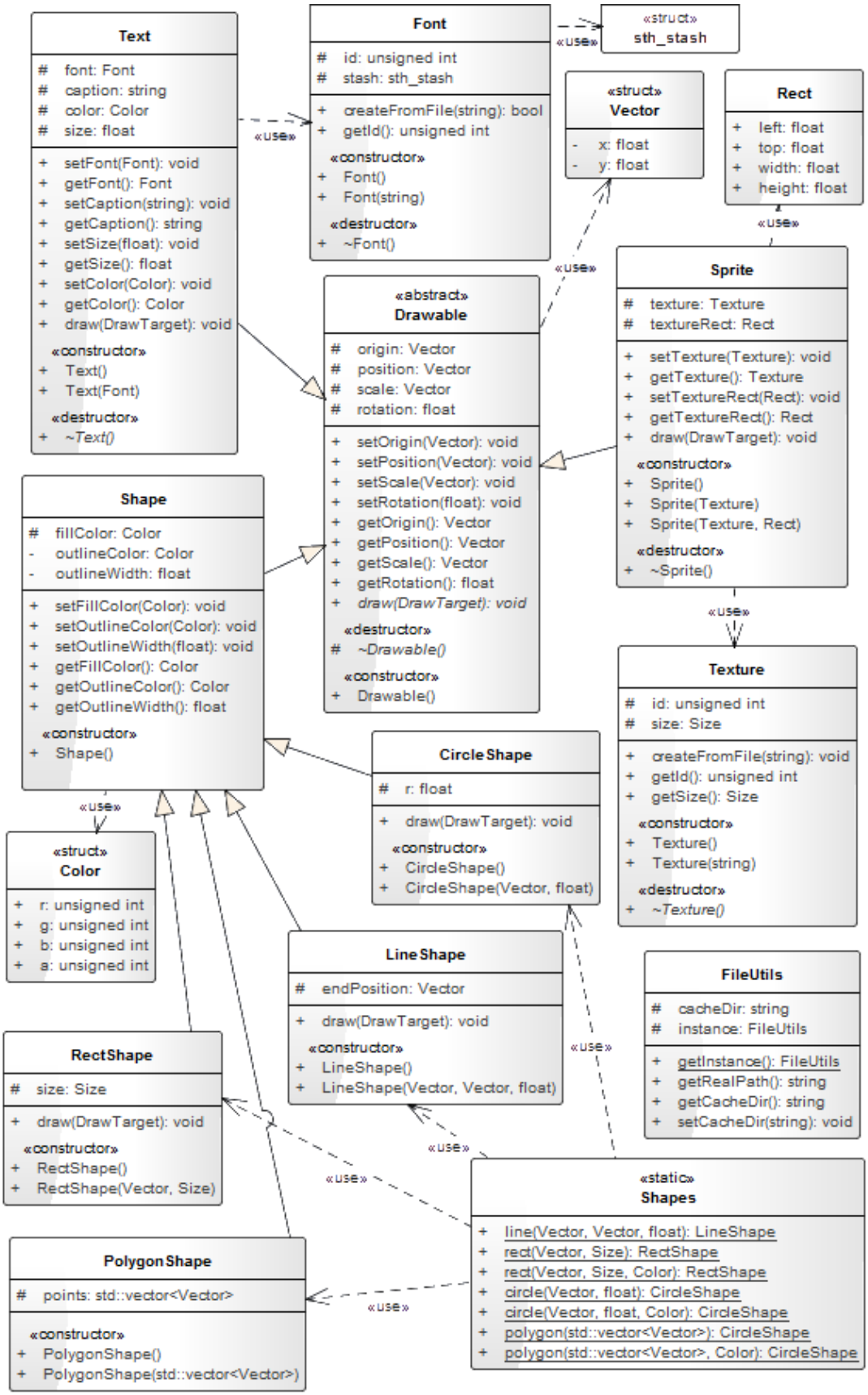
Terdapat beberapa pilihan *Application Programming Interface* (API) bagi pengembang yang ingin membangun aplikasi grafis, salah satunya adalah OpenGL. Namun, karena OpenGL merupakan API tingkat rendah, untuk melakukan sesuatu yang *trivial* sekalipun pengembang harus memanggil cukup banyak fungsi-fungsi pra-operasi yang relatif kompleks. Hal tersebut membuat proses pengembangan aplikasi itu sendiri menjadi suatu aktivitas yang rumit dan memiliki kurva pembelajaran yang relatif tinggi. Dalam penelitian ini dibahas bagaimana sebuah lapisan abstraksi di atas API OpenGL dengan memanfaatkan pola-pola desain tertentu dapat dibangun untuk mengurangi kompleksitas kode dan dapat dijalankan di platform desktop dan mobile. Penelitian ini menerapkan beberapa pola desain seperti *Façade*, *Template Method Pattern*, dan *Observer* untuk membangun sebuah library yang mengenkapsulasi sebagian kecil fungsionalitas 2-dimensi dari OpenGL ES. Hasil dari penelitian ini diharapkan dapat memberikan kemudahan bagi pengembang yang ingin mengembangkan perangkat lunak grafis untuk platform desktop dan mobile.

2. Gambaran Umum *Library*

Hasil penelitian ini berupa sebuah *library* yang meliputi abstraksi sistem *windowing* pada sistem operasi yang didukung serta fitur dasar untuk penggambaran grafik 2-dimensi. Untuk sistem *windowing*, terdapat kelas *WindowProtocol* yang menjadi kelas dasar dari kelas *Window* yang memiliki implementasi berbeda pada masing-masing sistem operasi yang didukung. Untuk grafik 2-dimensi, terdapat kelas *Texture* dan *Sprite* untuk menampilkan gambar, kelas *Font* dan *Text* untuk menampilkan teks, dan kelas *Shapes* sebagai pabrik abstrak bentuk-bentuk primitif. Selain itu, terdapat juga beberapa kelas, struktur, atau tipe data lain sebagai pendukung.

3. Diagram Kelas




Text
font: Font
caption: string
color: Color
size: float
+ setFont(Font): void
+ getFont(): Font
+ setCaption(string): void
+ getCaption(): string
+ setSize(float): void
+ getSize(): float
+ setColor(Color): void
+ getColor(): Color
+ draw(DrawTarget): void
«constructor»
+ Text()
+ Text(Font)
«destructor»
+ ~Text()
«use»
Font
id: unsigned int
stash: sth_stash
+ createFromFile(string): bool
+ getId(): unsigned int
«constructor»
+ Font()
+ Font(string)
«destructor»
+ ~Font()
«use»
«struct»
sth_stash
«struct»
Vector
- x: float
- y: float
Rect
+ left: float
+ top: float
+ width: float
+ height: float
«use»
«use»
«abstract»
Drawable
origin: Vector
position: Vector
scale: Vector
rotation: float
+ setOrigin(Vector): void
+ setPosition(Vector): void
+ setScale(Vector): void
+ setRotation(float): void
+ getOrigin(): Vector
+ getPosition(): Vector
+ getScale(): Vector
+ getRotation(): float
+ draw(DrawTarget): void
«destructor»
~Drawable()
«constructor»
+ Drawable()
Sprite
texture: Texture
textureRect: Rect
+ setTexture(Texture): void
+ getTexture(): Texture
+ setTextureRect(Rect): void
+ getTextureRect(): Rect
+ draw(DrawTarget): void
«constructor»
+ Sprite()
+ Sprite(Texture)
+ Sprite(Texture, Rect)
«destructor»
+ ~Sprite()
«use»
Shape
fillColor: Color
- outlineColor: Color
- outlineWidth: float
+ setFillColor(Color): void
+ setOutlineColor(Color): void
+ setOutlineWidth(float): void
+ getFillColor(): Color
+ getOutlineColor(): Color
+ getOutlineWidth(): float
«constructor»
+ Shape()
Texture
id: unsigned int
size: Size
+ createFromFile(string): void
+ getId(): unsigned int
+ getSize(): Size
«constructor»
+ Texture()
+ Texture(string)
«destructor»
+ ~Texture()
«use»
CircleShape
r: float
+ draw(DrawTarget): void
«constructor»
+ CircleShape()
+ CircleShape(Vector, float)
«use»
«struct»
Color
+ r: unsigned int
+ g: unsigned int
+ b: unsigned int
+ a: unsigned int
LineShape
endPosition: Vector
+ draw(DrawTarget): void
«constructor»
+ LineShape()
+ LineShape(Vector, Vector, float)
«use»
FileUtils
cacheDir: string
instance: FileUtils
+ getInstance(): FileUtils
+ getRealPath(): string
+ getCacheDir(): string
+ setCacheDir(string): void
RectShape
size: Size
+ draw(DrawTarget): void
«constructor»
+ RectShape()
+ RectShape(Vector, Size)
«use»
«use»
«static»
Shapes
+ line(Vector, Vector, float): LineShape
+ rect(Vector, Size): RectShape
+ rect(Vector, Size, Color): RectShape
+ circle(Vector, float): CircleShape
+ circle(Vector, float, Color): CircleShape
+ polygon(std::vector<Vector>): CircleShape
+ polygon(std::vector<Vector>, Color): CircleShape
PolygonShape
points: std::vector<Vector>
«constructor»
+ PolygonShape()
+ PolygonShape(std::vector<Vector>)
«use»

4. Pola Desain Yang Diterapkan

1. Pabrik Abstrak (*Abstract factory*)

Antarmuka yang dapat membuat suatu objek yang diperlukan tanpa memanggil kelas yang sebenarnya. Pola ini dimanfaatkan untuk mengimplementasikan kelas penggambar bentuk primitif.

```
Shapes::circle(Vector(760, 330), 50, Color::Orange);
```

2. Resource acquisition is initialization (RAII)

Teknik untuk memastikan bahwa suatu objek akan terhapus dengan benar ketika sudah tidak perlukan lagi dengan cara menginstansiasikan objeknya di dalam objek lain yang sudah dipastikan memiliki jangka hidup yang lebih lama dan didealokasikan pada saat destruksi dari objek yang bertanggungjawab.

3. Singleton

Berupa kelas yang pasti hanya memiliki satu instansi objek dan menyediakan sebuah titik yang dapat diakses oleh siapapun untuk mengakses metode-metode yang terdapat pada instansi kelas tersebut.

```
FileUtils::getInstance()->getRealPath("background.png")
```

4. Façade

Pola desain Façade mengenkapsulasi fungsi tingkat rendah dan struktur data ke dalam antarmuka kelas berorientasi objek yang lebih singkat, kuat, portabel, dan lebih mudah diperlihara. Pola ini diterapkan sebagai gambaran besar dari keseluruhan *library* yang mengenkapsulasi sebagian fungsionalitas OpenGL ES beserta struktur datanya dan sistem *windowing* sederhana dari sistem operasi yang didukung sehingga kompleksitas di belakangnya akan tertutupi.

5. Observer

Juga dikenal sebagai pola *publish*/*subscribe* atau pola *signal*/*slot* dimana metode-metode *slot* dapat meminta *signal* untuk memanggilnya ketika suatu keadaan tertentu di dalam sistem tersebut berubah. Pola ini akan diterapkan pada sistem pengelolaan *event*.

```
// Mendaftarkan fungsi listener
window.setEventFunction(processEvent);

window.start();
// Menerima dan memroses event
void processEvent(const Event &event)
{
	if(event.type == KeyDown || event.type == KeyUp)
	{
		std::string type = event.type == KeyDown ? "diangkat" :
"ditekan";

		printf("Tombol %s %s" + keyStr[event.key], type.c_str());
	}

}
```

6. Template Method

Template method merupakan sebuah pola tingkah laku yang menentukan kerangka dari suatu proses yang kemudian menurunkannya pada subkelas. Pola ini akan banyak digunakan untuk mendefinisikan kelas-kelas dasar yang selanjutnya akan diturunkan menjadi subkelas-subkelas yang memiliki fungsionalitas serupa namun dengan spesialisasi atau peruntukkan tertentu.

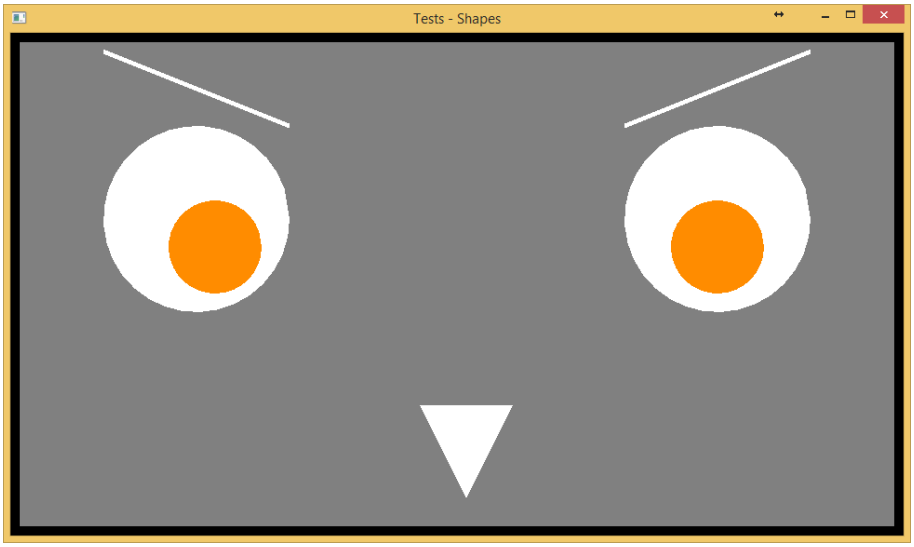
```
class Drawable;
class Sprite : public Drawable;
class Text : public Drawable;
class Shape : public Drawable;
class RectShape : public Shape;
class CircleShape : public Shape;
```

APLIKASI PENGUJIAN *LIBRARY* YANG TELAH DIKEMBANGKAN

***Source code* lengkap dari *library* dapat di-*clone* di** http://bitbucket.org/djzmo/cybil2d

1. MEMBUAT BENTUK PRIMITIF

```
#include <cybil/cybil.h>

using namespace cybil;

RectShape background;
PolygonShape poly;

void draw(const DrawTarget &target, double delta)
{
	target.draw(background);

	target.draw(Shapes::circle(Vector(200, 200), 100));
	target.draw(Shapes::line(Vector(100, 20), Vector(300, 100), 5));
	target.draw(Shapes::circle(Vector(760, 200), 100));
	target.draw(Shapes::line(Vector(860, 20), Vector(660, 100), 5));

	target.draw(Shapes::circle(Vector(220, 230), 50, Color::Orange));
	target.draw(Shapes::circle(Vector(760, 230), 50, Color::Orange));

	target.draw(poly);
}

int main(int argc, char **argv)
{
    WindowConfiguration conf("Tests - Shapes", cybil::Size(960, 540));
    Window window(conf);

    Vector points[] = { Vector(0, 0), Vector(100, 0), Vector(50, 100) };
    background = Shapes::rect(Vector(10, 10), Size(940, 520));
    background.setFillColor(Color::Gray);
    poly = Shapes::polygon(Vector(440, 400), std::vector<Vector>(points, points +
sizeof(points) / sizeof(points[0])));

    window.setDrawFunction(draw);
    window.setAcceptsMouseMovedEvents(true);
    window.start();

    return 0;
}
```

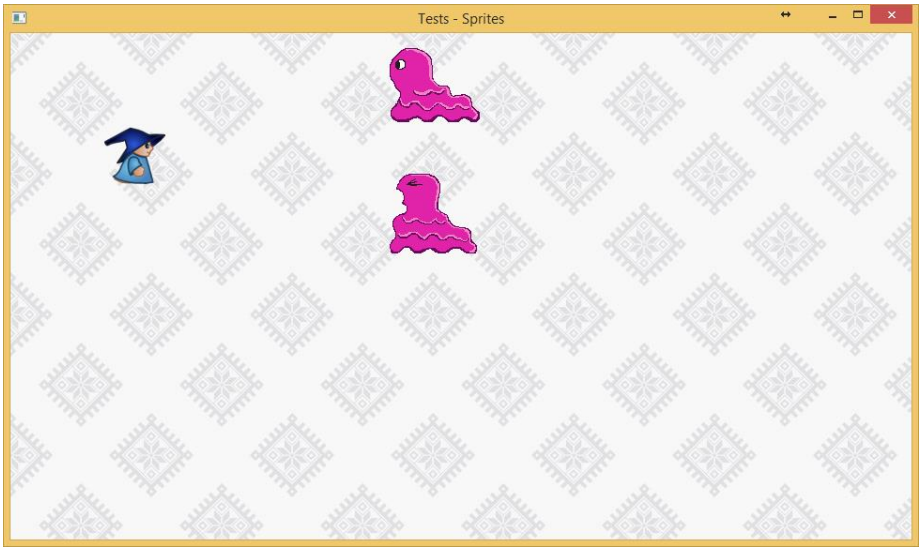


Gambar 1. Tampilan hasil aplikasi pengujian bentuk primitif

2. MENAMPILKAN *SPRITE*

```
#include <cybil/cybil.h>

using namespace cybil;

Texture sorloTex;
Texture gloopTex;
Texture backgroundTex; // http://subtlepatterns.com/eight-horns
Sprite background;
Sprite sorlo; // http://opengameart.org/content/sorlo-a-funny-sorcerer
Sprite gloop[2]; // http://opengameart.org/content/gloop-monster

void draw(const DrawTarget &target, double delta)
{
    target.draw(background);
    target.draw(sorlo);
    target.draw(gloop[0]);
    target.draw(gloop[1]);
}

int main(int argc, char **argv)
{
    WindowConfiguration conf("Tests - Sprites", cybil::Size(960, 540));
    Window window(conf);

    backgroundTex.createFromFile("background.png");
    sorloTex.createFromFile("sorlo.png");
    gloopTex.createFromFile("gloop.png");
    sorlo.setTexture(&sorloTex);
    sorlo.setPosition(Vector(100, 100));
    background.setTexture(&backgroundTex);
    gloop[0].setTexture(&gloopTex);
    gloop[0].setTextureRect(Rect(0, 0, 100, 85));
    gloop[0].setPosition(Vector(400, 10));
```

```
    gloop[1].setTexture(&gloopTex);
    gloop[1].setTextureRect(Rect(300, 85, 100, 85));
    gloop[1].setPosition(Vector(400, 150));

    window.setDrawFunction(draw);
    window.setAcceptsMouseMovedEvents(true);
    window.start();

    return 0;
}
```



Gambar 2. Tampilan hasil aplikasi pengujian *sprite*

3. MENAMPILKAN TEKS

```
#include <cybil/cybil.h>

using namespace cybil;

Font font;
Text texts[5];

void draw(const DrawTarget &target, double delta)
{
    for(int i = 0; i < 5; i++)
        target.draw(texts[i]);
}

int main(int argc, char **argv)
{
    WindowConfiguration conf("Tests - Texts", cybil::Size(960, 540));
    Window window(conf);

    font.createFromFile("Bebas.ttf");
```

```
    char caption[64];
    for(int i = 0; i < 5; i++)
    {
       sprintf(caption, "Hello world %d", i);
       texts[i].setCaption(caption);
       texts[i].setFont(&font);
       texts[i].setSize(40 + i * 10);
       texts[i].setPosition(Vector(10, 20 + i * 75));
    }

    texts[0].setColor(Color::Red);
    texts[1].setColor(Color::Green);
    texts[2].setColor(Color::Blue);
    texts[3].setColor(Color::Orange);
    texts[4].setColor(Color::Purple);

    window.setDrawFunction(draw)
    window.setAcceptsMouseMovedEvents(true);
    window.start();

    return 0;
}
```

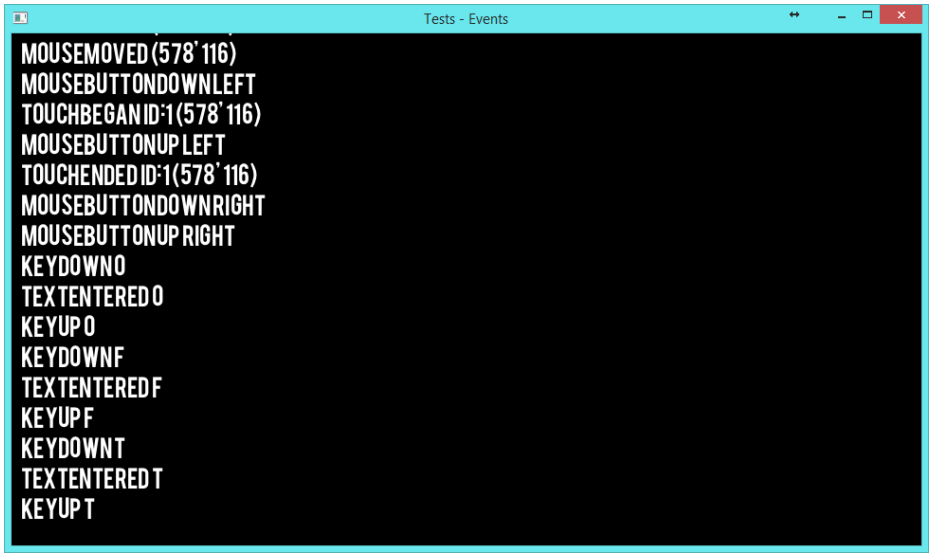
Gambar 3. Tampilan hasil aplikasi pengujian teks

4. MENANGANI *EVENT*

```
#include <cybil/cybil.h>
#include <string>
#include <algorithm>

using namespace cybil;

std::string debugStr;
Text debugText;
Font defaultFont;
```

```
char *keyStr[] =
{
    "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M",
    "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z",
    "Number0", "Number1", "Number2", "Number3", "Number4",
    "Number5", "Number6", "Number7", "Number8", "Number9",
    "Numpad0", "Numpad1", "Numpad2", "Numpad3", "Numpad4",
    "Numpad5", "Numpad6", "Numpad7", "Numpad8", "Numpad9",
    "F1", "F2", "F3", "F4", "F5", "F6",
    "F7", "F8", "F9", "F10", "F11", "F12",
    "UpArrow", "LeftArrow", "DownArrow", "RightArrow", "PageUp", "PageDown",
    "LShift", "RShift", "LAlt", "RAlt", "LCtrl", "RCtrl",
    "Tab", "End", "Home", "Delete", "CapsLock", "System",
    "Insert", "Space", "BackSpace", "Enter", "Escape",
    "Pause", "Menu", "Add", "Divide", "Subtract", "Multiply",
    "Comma", "Period", "Slash", "BackSlash", "SemiColon",
    "SingleQuote", "LBracket", "RBracket", "Tilde",
    "Dash", "Equal", "PrintScreen"
};

void appendLog(const std::string &text)
{
       if(std::count(debugStr.begin(), debugStr.end(), '\n') < 16)
              debugStr += "\n" + text;
       else debugStr = debugStr.substr(debugStr.find('\n') + 1) + "\n" + text;
       printf("%s\n", text.c_str());
       debugText.setCaption(debugStr);
}

void draw(const DrawTarget &target, double delta)
{
       target.draw(debugText);
}

void processEvent(const Event &event)
{
       char buffer[64];
       if(event.type == KeyDown || event.type == KeyUp)
       {
              std::string type = event.type == KeyDown ? "KeyDown" : "KeyUp";

              appendLog(type + " " + keyStr[event.key]);
       }
       else if(event.type == TextEntered)
              appendLog(std::string("TextEntered ") + (char)event.charCode);
       else if(event.type == MouseButtonDown || event.type == MouseButtonUp)
       {
              std::string type = event.type == MouseButtonDown ? "MouseButtonDown" :
"MouseButtonUp";
              std::string button = "Middle";

              if(event.mouseButton == Left)
                     button = "Left";
              else if(event.mouseButton == Right)
                     button = "Right";

              appendLog(type + " " + button);
       }
```

```
	else if(event.type == MouseMoved)
	{
		sprintf(buffer, "MouseMoved (%d, %d)", (int)event.mousePosition.x,
(int)event.mousePosition.y);
		appendLog(buffer);
	}
	else if(event.type == MouseWheelScrolled)
	{
		sprintf(buffer, "MouseWheelScrolled (%d)", event.mouseWheelDelta);
		appendLog(buffer);
	}
	else if(event.type == TouchBegan || event.type == TouchMoved || event.type ==
TouchEnded)
	{
		std::string type = "TouchMoved";

		if(event.type == TouchBegan)
			type = "TouchBegan";
		else if(event.type == TouchEnded)
			type = "TouchEnded";
		sprintf(buffer, "%s ID:%d (%d, %d)", type.c_str(), event.touchId,
(int)event.touchPosition.x, (int)event.touchPosition.y);
		appendLog(buffer);
	}
	else if(event.type == WindowResized)
	{
		sprintf(buffer, "WindowResized (%d, %d)", (int)event.windowSize.width,
(int)event.windowSize.height);
		appendLog(buffer);
	}
}

int main(int argc, char **argv)
{
    WindowConfiguration conf("Tests - Events", cybil::Size(960, 540));
    Window window(conf);

    defaultFont.createFromFile("Bebas.ttf");
    debugText.setFont(&defaultFont);
    debugText.setPosition(Vector(10, 10));
    debugText.setSize(32);

    window.setDrawFunction(draw);
    window.setEventFunction(processEvent);
    window.start();

    return 0;
}
```



Gambar 4. Tampilan hasil aplikasi pengujian *event*