# Benutzerdokumentation

Ich habe eine zweidimensionale geometrische Bibliothek entwerft, damit der Benutzer einfach geometrische Operationen durchführen kann. In dieser Dokumentation werde ich zeigen mit Beispielen, wie kann es benutzt werden und worum soll man achten, falls man es benutzt.

Diese Bibliothek ist nur mit OpenGL nutzbar, deshalb soll man GLUT (OpenGL Utility Toolkit) oder andere Bibliothek benutzen, die OpenGL enthaltet. Aber ich werde es mit GLUT zeigen, weil damit es einfach zu verstehen ist. Ich werde nur die Grundfunktionen von GLUT erklären, aber es hat mehr Verbrauchsmöglichkeiten.

## Ein bisschen über GLUT

Sehen wir welche Funktionen brauchen wir von GLUT, und was können wir dort tun.

```
glutCreateWindow ("GEOM 2D Test Program");
```

Mit dieser Funktion kann man ein Fenster erzeugen. Die gegebene Zeichenkette wird das Titel des Fensters.

```
glutInitWindowSize (width, height);
```

Mit dieser Funktion kann man die Größe des Fensters eingeben.

```
glutFullScreen ();
```

Mit dieser Funktion kann man das Fenster zu Vollbild einstellen.

```
glutDisplayFunc (display);
```

Mit dieser Funktion kann man eine Funktion eingeben, die anrufen wird, falls die Oberfläche wiederzeichnet werden muss.

```
glutMouseFunc (mouse_func);
glutPassiveMotionFunc (passive_motion_func);
glutMotionFunc (motion_func);
glutKeyboardFunc (keyboard_func);
```

Mit diesen Funktionen kann man Funktionen eingeben, die anrufen wird, falls verschiedene Event auftritt. Zum Beispiel: klicken oder Taste drücken.

```
glutMainLoop ();
```

Mit dieser Funktion kann das Main Loop beginnen. Diese Funktion sollte man zu letztes Mal anrufen, weil irgendwas steht nach dem Anruf, wird nicht durchgeführt.

GLUT hat leider keine solche Funktion, damit wir einstellen kann, was sollte am Ende anrufen, deshalb man soll *'atexit(func)'* benutzen, falls man dynamische Speicherung benutzt, und möchtet man beim Schließen befreien. Es ist in **<stdlib.h>** definiert.

## Initialisierung der Objekte

Man kann die Objekte durch Konstruktoren initialisieren. Die Komponenten haben sehr eindeutige Definitionen. Ein paar Beispiele:

```
Point p(59, 34);
Point* a = new Point(0, 100);
Point* b = new Point(100, 100);
Point* c = new Point(159, 100);
Segment* segment = new Segment(p, Point(50, 50));
MyPolygon* poly = new MyPolygon (a, b, c);
Rect* rect = new Rect (1, 1, 50, 50);
```

Die Konstruktoren kann man mit Referenz und auch mit Zeiger anrufen, Damit der Benutzer soll nicht so viel Adresse-Operationen (& und *) benutzen.

## Benutzung der Objekte

Es ist auch merkbar, wenn man eine Komponente auszeichnen möchte, sollte er es dynamisch definieren. Damit kann man es zu einer Heterogen-Kollektion hinzufügen und wird die 'display' Funktion einfacher und schneller. Zum Beispiel:

```
//Heterogen-Kollektion
vector<Drawable*> drawables;


//Initialisieren
Rect* rect = new Rect (1, 1, 50, 50);
drawables.push_back (rect);


//Display function
for ( int i = 0; i < drawables.size (); i++ ) {
   drawables[i]->draw ();
}


//Clean function beim Exit
for ( int i = 0; i < drawables.size (); i++ ) {
   delete drawables[i];
}
```

Man kann es auch mit Transformable*, Area* oder Shape* verwendet, damit werden mehrere Funktionen nutzbar beim Iterationen. Zum Beispiel falls wir möchten jede Tranformable Komponente auf einen Punkt spiegeln, dann:

```
//Heterogen-Kollektion
vector<Transformable*> transformables;
//Die Punkt, darauf wie spiegeln
Point* point_mirror;


//Initialisieren
Segment* segment = new Segment (1, 1, 50, 50);
transformables.push_back (rect);
point_mirror = new Point (55, 52);
transformables.push_back (point_mirror);


//Display function
for ( int i = 0; i < transformables.size (); i++ ) {
   transformables [i]->draw ();

   transformables[i]->reflect (point_mirror);
   transformables[i]->draw ();
   transformables[i]->reflect (point_mirror);
}


//Clean beim Exit
for ( int i = 0; i < transformables.size (); i++ ) {
   delete transformables [i];
}
```

Falls wir möchten den Punkt nicht auszeichnen, dann sollte man es nicht zu der Kollektion fügen, aber dann man muss achten, dass es soll beim ’Clean’ separiert befreien.

## Benutzung bei den Events

Mann soll aufmerksam sein, wenn man die Event Funktionen von GLUT benutzt. Die y-Achse ist umgekehrt in diesem fall, deshalb zu erste soll man es zurückkehren. und falls wir brauchen die Oberfläche zu wiederzeichnen, dann soll ‚display‘ Funktion anrufen, sondern werden wir die Veränderung nicht sehen.

```
void motion_func (int x, int y) {

   y = HEIGHT - y;

   if ( rect->isInside (&p)) {
      rect->setColor (1.0f, 0.0f, 0.0f);
   } else {
      rect->setColor (1.0f, 1.0f, 1.0f);
   }

   display ();
}
```

Diese Funktion macht unser Rechteck rot, falls die Maus ist innerhalb des Rechtecks, sonst ist es weiß.

Mehrere Beispiele und Benutzung kann man in dem Testprogram anschauen, und die volle Funktionalität befindet sich in die Kodierungsdokumentation.