

### Aufgabe 5: “mini-Qt in OpenGL”

Alexander Kramer

---

## IMPLEMENTIERUNG

Die Namensgebung ist an die des Qt angelehnt. Alle Klassennamen beginnen mit einem Q.

Die zentrale Klasse ist die QWidget Klasse. Diese Klasse bietet die Basisfunktionalität für alle abgeleiteten Klassen. Ein Widget definiert einen rechteckigen Bereich innerhalb eines anderen Widgets. Ein Widget ist für seine Darstellung auf dem Bildschirm verantwortlich und ist zuständig für die Mausinteraktion, die innerhalb seines Bereiches stattfindet.

Alle GUI Widgets bilden einen GUI-Baum wobei die Widgets selbst die Knoten darstellen. Dazu speichert die QWidget Klasse in einer Liste Referenzen auf die Kinder-Knoten. Das sind ihr untergeordnete Widgets (im Folgenden auch als Children bezeichnet). Und sie speichert ebenfalls eine Referenz auf übergeordnetes Widget, den Vater-Knoten (oder engl. Owner). Jedes Widget hat also einen Owner, mit Ausnahme von dem Widget, welches sich in der Wurzel dieses Baumes befindet. Das ist in den meisten Fällen ein QScreen Widget, welches gleichzeitig die Schnittstelle zu den GLUT Callbacks herstellt. Ein Widget zeichnet sich selbst in den vorbereiteten graphischen Kontext und geht dabei davon aus, dass der Koordinaten-Ursprung in seiner oberen linken Ecke liegt. Dazu wird der Matrizen Stack der OpenGL ausgenutzt. Beim Zeichnen wird der GUI-Baum traversiert und jeder Knoten liegt eine Translationsmatrix auf den Stack, bevor er sich selbst zeichnet und stellt die Matrix wieder her nachdem alle untergeordnete Knoten rekursiv traversiert wurden. Beim verarbeiten von Maus Ereignissen werden die Mauskoordinaten ebenfalls zwischengespeichert, bevor sie relativiert und rekursiv weitergegeben werden.

Die Tastatur Ereignisse werden entlang dem s.g. Fokus-Pfad weitergeleitet. Der Fokus Pfad ist ein Pfad im GUI-Baum von der Wurzel des Baums bis zu einem Blatt-Knoten, der den Fokus für sich angefordert hat. In der Regel wird der Fokus angefordert, sobald eine Maus-Taste gedrückt wurde.

Beim Löschen eines Widgets werden die Children-Widgets ebenfalls rekursiv gelöscht. Man darf die Widgets nicht mit delete Anweisung löschen sondern mit der Methode free() die dafür sorgt, dass alle untergeordnete Widgets gelöscht werden. Das Löschen selbst geschieht nicht sofort - dies würde die Baumtraversierung zum Absturz bringen, da man sich zu diesem Zeitpunkt Mitten in der Traversierung des Baumes befindet. Die zu löschende Widgets werden im GUI-Baum nach oben weitergereicht und schliesslich in einer Liste des QScreen Objekts abgelegt. Vor dem Senden des nächsten Events wird diese Liste überprüft und die Widgets entfernt, bevor das Event weitergereicht wird.

Die Klasse QWidget empfängt alle Events über die event(QEvent\* e) Methode. Je nach Typ vom Event werden verschiedene Methoden des Widgets aufgerufen(draw,mouseEvent,keyboardEvent). In der Methode draw() zeichnet sich QWidget mit Hilfe von OpenGL selbst.

Die Klasse QScreen bekommt die Maus/Tastatur Events von GLUT. Sie besitzt ihre eigene Handler-Methode für die Callback Routinen von GLUT. (glutMouse, glutKeyboard usw.) Diese Routinen müssen aus dem Hauptprogramm aufgerufen werden.

Die Ereignisbehandlungsroutinen werden als Zeiger auf Funktionen implementiert. Es ist damit möglich eine einzige Ereignisbehandlungsroutine mit mehreren Widgets zu verknüpfen. Leider ist dieser Ansatz nicht so mächtig wie signal/slot Mechanismus, er ist dafür besonders einfach in der Implementierung und Verständnis.

## MANUAL

Um die miniQt Bibliothek in vorhandenen OpenGL/GLUT Projekte zu benutzen muss man eine Instanz der QScreen Klasse erstellen und dafür sorgen, dass innerhalb der GLUT Callbacks die entsprechenden Methoden der QScreen Klasse aufgerufen werden und dass die Maus-Koordinaten, die an QScreen übergeben werden, korrekt sind.

Beispiel einer möglichen Einbettung von QScreen:

```
QScreen *screen;
```

```
void glutResize(int width, int height)
```

```
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0,width,height,0,-1,1);
    glMatrixMode(GL_MODELVIEW);
    screen->setWidth(width);
    screen->setHeight(height);
}

void glutKeyboard(unsigned char key, int x, int y)
{
    screen->glutKeyboard(key,x,y);
}

void glutKeyboardUp(unsigned char key, int x, int y)
{
    screen->glutKeyboardUp(key,x,y);
}

void glutSpecial(int key, int x, int y)
{
    screen->glutSpecialFunc(key,x,y);
}

void glutMotion(int x, int y)
{
    screen->glutMotion(x,y);
}

void glutMouse(int button, int state, int x, int y)
{
    screen->glutMouse(button,state,x,y);
}

void glutDisplay()
{
```

```
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    screen->glutDisplay();
    glutSwapBuffers();
}
```

```
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);

    glutInitWindowSize(500, 400);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("mini QT");

    screen = new QScreen(0);

    // hier Fenster mit Widgets erstellen

    QWindow * win = new QWindow(screen);
    win->setWidth(100);
    win->setHeight(100);

    glutReshapeFunc(glutResize);
    glutDisplayFunc(glutDisplay);
    glutKeyboardFunc(glutKeyboard);
    glutKeyboardUpFunc(glutKeyboardUp);
    glutSpecialFunc(glutSpecial);
    glutMouseFunc(glutMouse);
    glutMotionFunc(glutMotion);

    glutMainLoop();
    return 0;
}
```

## 0.1 Hinzufügen von Fenstern und Widgets

Die Zugehörigkeit von Widgets und Fenstern (Fenster sind selbst Widgets) wird über den Parameter im Konstruktor festgelegt. Wir müssen dem Konstruktor eines Fensters eine Screen Referenz übergeben.

Wir können obiges Beispiel wie folgt erweitern, um ein Fenster zu erstellen:

```
// hier Fenster mit Widgets erstellen
QWindow * win = new QWindow(screen);
win->setWidth(100);
win->setHeight(100);
win->setTop(10);
win->setLeft(10);
```

Damit erstellen wir ein Fenster mit Breite und Höhe gleich 100 Pixel. Natürlich wollen wir auch verschiedene Widgets in unserem Fenster haben, z.B. einen Button. Dazu müssen wir lediglich eine Zeile hinzufügen:

```
// hier Fenster mit Widgets erstellen
QWindow * win = new QWindow(screen);
win->setWidth(100);
win->setHeight(100);
win->setTop(10);
win->setLeft(10);

// einen Button hinzufügen
QCaptionButton * bn = new QCaptionButton(win);
```

Im Konstruktor von `QCaptionButton` haben wir unser Fenster angegeben.

## 0.2 Festlegen einer Ereignisbehandlungsroutine

Zuerst fügen wir vor der `main()` Methode unsere Routine ein:

```
void myMethod(QWidget* Sender, QEvent* e)
{
```

```
QCaptionButton* bn = static_cast<QCaptionButton*>(Sender);  
if (bn)  
    cout << "Sie haben " << bn->getCaption() << " angeklickt!" << endl;  
}
```

Jetzt verknüpfen wir den onPress Event des Buttons mit gerade erstellter Methode:

```
// hier Fenster mit Widgets erstellen  
QWindow * win = new QWindow(screen);  
win->setWidth(100);  
win->setHeight(100);  
win->setTop(10);  
win->setLeft(10);  
  
// einen Button hinzufügen  
QCaptionButton * bn = new QCaptionButton(win);  
bn->setCaption("Hello");  
// Ereignisbehandlungsroutine setzen  
bn->onPress = myMethod;
```

Wenn wir auf den Button klicken, erscheint in der Konsole:

Sie haben Hello angeklickt!