Schritt 1: Ein Fenster erzeugen & OpenGL initialisieren.

Wenn wir OpenGL programmieren, brauchen wir als allererstes einen sogenannten OpenGL Context. Die Erstellung eines solchen ist vom Betriebssystem abhängig, mühsam, und führt nicht unbedingt zu einem Erkenntnisgewinn, sprich: Es ist eigentlich etwas, das man nicht selbst tun möchte. In Vertiefung CG benutzen wir hierfür eine 3D Party Library, SFML, welche das Erstellen des Contexts – und eines Fensters zum Anzeigen unserer Grafik – übernimmt.

Zusätzlich verwenden wir die Library GLEW. GLEW übernimmt die dankenswerte Aufgabe, die OpenGL Funktionspointer aus der auf Ihrem Rechner installierten Treiber DLL abzugreifen und sie dann sauber und schön mit dem Präfix *gl-* zur Verfügung zu stellen. Wer ganz genau wissen möchte warum und wieso, lese folgende Erklärung zu dem Thema. Wie schon bei der OpenGL Context Erstellung gilt folgendes: Das ist extrem mühsam, es bringt einem kaum Erkenntnisse, und man will es wirklich nicht selber programmieren müssen. Praktisch jeder Programmierer, der unter Windows und C++ mit OpenGL arbeitet, verwendet GLEW oder alternativ GLEE.

Damit kommen wir zur Basisanwendung. Jedes Sample und jedes Hausübung wird folgenden Grundaufbau haben:

```
int main( int argc, char** argv )
{
	// WINDOW CREATION, ALSO CREATES OPENGL CONTEXT & INITIALIZES GLEW
	sf::Window window( sf::VideoMode( 800, 800, 32 ), "tutorial", sf::Style::Close,
sf::ContextSettings( 16, 0, 4, 3, 3 ) );
	if ( glewInit() != GLEW_OK ) return 0;

	// INIT: ALLOCATE OPENGL RESOURCES

	// MAIN LOOP BEGINS
	while ( true )
	{
		bool exit = false;
		sf::Event ev;
		// POLL WINDOW EVENTS -> HANDLE USER INPUT
		while ( window.pollEvent( ev ) )
		{
			if ( sf::Event::Closed == ev.type )
			{
				exit = true;
				break;
			}
		}
		if ( exit ) break;

		// RENDER: MAKE OPENGL CALLS, START WORK @GPU

		// UPDATE: PERFORM CALCULATIONS @CPU FOR NEXT FRAME

		// DISPLAY: SYNC WITH GPU, UPDATE WINDOW CONTENT
		window.display();
	}

	// UN-INIT: RELEASE ALL RESOURCES HERE

	return 0;
}
```

Die im Code vorhandenen Kommentare markieren wichtige Stellen im Programmfluss. In der CG1 war Ihr OpenGL Code immer in eine *App* eingebettet, und Sie hatten nie die völlige Kontrolle über ihr Programm: In der `Main()` haben Sie typischerweise die `Run()` Funktion ihrer App aufgerufen, worauf diese sich dann um alles Weitere gekümmert hat. Sie selber haben Funktionen wie `OnRenderFrame`, `OnUpdateFrame`, `OnLoad`, `OnUnload` usw. implementiert, welche zu bestimmten Zeiten automatisch aufgerufen wurden. Dies ist ein beliebter Ansatz, auch im Sinne der Objektorientierung... die SFML ist eine der ganz wenigen Libraries ihrer Art, die diesem Modell nicht folgt: Ich möchte, dass unsere Codesamples so transparent wie möglich sind.

Falls Sie sich mit dem App Ausführungsmodell aus der CG1 wohler fühlen, ist es übrigens recht einfach, so etwas für die SFML zu bauen. Es wäre auch grundsätzlich eine gute Übung für Sie...

Zur Erklärung der einzelnen Kommentare im Code

WINDOW CREATION: Wir starten immer damit, ein Fenster - `sf::Window` - mit einer bestimmten Größe anzulegen. (Weitere Settings im Sample Code: 16 Bit Z-Buffer, 0 Bit Stencilbuffer, 4-faches Multisampling, OpenGL Version 3.3, Compatibility Modus) Wenn diese Erstellung gelingt, existiert auch sofort ein valider OpenGL Context, d.h. erst dann kann man OpenGL Befehle überhaupt aufrufen (davor → Absturz). Sofort nachdem der Context da ist, wird auch gleich die GLEW initialisiert – und sollte dies fehlschlagen, ist im Grunde eh alles Weitere sinnlos, wir können dann nämlich nichts rendern. Wir brechen die Anwendung in dem Fall auf der Stelle ab. Zur WINDOW CREATION ist weiter nichts zu tun, außer evtl. die Settings des Fensters (Größe, Titel) zu ändern.

Als nächstes kommen wir in die INIT Phase unserer Anwendung, das beinhaltet z.B.:

- Laden der GLSL Shader, Kompilieren selbiger und Linken eines oder mehrerer GLSL Programmobjekte
- Laden von Bildern und 3D Modellen & Upload Selbiger auf die Grafikkarte, in Form von Textures und (Vertex) Buffer Objects, sowie das Initialisieren von Vertex Array Objects
- Erzeugen von Framebuffer Objects / Transform Feedbacks

Also im Grunde das Laden aller statischen Daten & Upload derselben auf die Grafikkarte, sowie Allokation von GPU Ressourcen.

Nach dem INIT gehen wir in den sogenannten MAIN LOOP, den wir durchführen, bis der Benutzer die Anwendung schließt. Der MAIN LOOP beinhaltet 4 Schritte:

HANDLE USER INPUTS: Wir setzen und mit dem User Input auseinander: Maus Events / Keyboard Inputs / Wurde die Anwendung geschlossen? / Wurde die Anwendung verkleinert oder vergrößert?

RENDER: Wir sagen der Grafikkarte, was sie für dieses Frame genau tun soll: Zeichnen der Geometrie / Postprocessing / Schatten berechnen / ein Partikelsystem im Shader updaten...

UPDATE: Wir führen notwendige Updates auf der CPU durch, während die Grafikkarte noch beschäftigt ist. (Anmerkung: OpenGL Befehle sind im Allgemeinen nicht blockierend, d.h. wir wissen nicht, wann genau die gewünschte Aktion durchgeführt wird, wenn wir zum Beispiel eine Zeichenfunktion aufrufen…) Anwendungsfall: All jene von Ihnen, die ein Partikelsystem *auf der CPU* programmiert haben, hätten im UPDATE die Partikelpositionen neu berechnet.

DISPLAY: Wie rufen die Funktion `sf::Window::display()` auf. Dies erzeugt einen Synchronisationspunkt mit der Grafikkarte (Wir warten, bis die Grafikkarte fertig mit Rendern ist) und stellt danach das Ergebnis im Fenster dar.

Mit dem Ausstieg aus dem MAIN LOOP (→ weil Fenster geschlossen) wird die Anwendung beendet. Das letzte, was für uns an dieser Stelle noch zu tun bleibt, ist das Freigeben aller OpenGL Ressourcen (UN-INIT). Immer, wenn Sie OpenGL Objekte mit `glGenX` anlegen, müssen Sie selbige später mit `glDeleteX` wieder freigeben. Dies betrifft Framebuffer Objects, Vertex Buffer Objects, Transform Feedbacks, Textures und Vertex Array Objects gleichermaßen.