UNIVERSITÄT OSNABRÜCK

Dr. Thomas Wiemann

8. Übungsblatt zur Vorlesung „Einführung in die Programmiersprache C++“

Wintersemester 2018 / 2019

Aufgabe 1: Container und Algorithmen (15 Punkte)

In dieser Aufgabe beginnen Sie, den Umgang mit der Standardbibliothek zu erlernen. Sie werden zum "Aufwärmen" einige kleinere Programme implementieren. Schreiben Sie für jede Teilaufgabe eine Datei `Aufgabe2_x.cpp` im Ordner aufgabe2. Schreiben Sie eine `CMakeLists.txt` zum Übersetzen der Programme. Benutzen Sie zum Lösen der Aufgaben möglichst nur STL-Container und Algorithmen.

1. Erstellen Sie einen `std::vector` mit 100 Integerwerten. Füllen Sie den Vektor mit 100 zufällig gewählten Werten aus dem Bereich 1 bis 100 durch einen geeigneten Generator. Generieren Sie die Zahlen durch das Übergeben eines Funktionsobjekts. Verwenden Sie den `copy()`-Algorithmus, um den Vektor auf den `cout`-Stream zu schreiben. Verwenden Sie dazu einen für Integer instanziierten `ostream_iterator`.
2. Erstellen Sie einen neuen Vektor, der 100 Strings enthält. Benutzen Sie einen anderen Generator, der die Strings mit zufälligen Zeichen füllt. Die erzeugten Zeichenketten sollen zwischen 5 und 10 Zeichen lang sein. Sortieren Sie den Vektor mit Hilfe des `sort()`-Algorithmus. Zeigen Sie das Ergebnis in gleicher Weise wie das aus Aufgabe 2.1 an.
3. Schreiben Sie ein einfaches Objekt, das ein `unary_function`-Modell implementiert. Das Objekt sollte einen internen Zustand haben, den man auf jedes Element des Vektors mit Integerwerten anwenden kann. Extrahieren Sie die Summe des Vektors und verwenden Sie dabei das Konstrukt `for_each`. Geben Sie die Summe auf dem Bildschirm aus!

Aufgabe 2: Factories für Meshes und Bitmaps (50 Punkte)

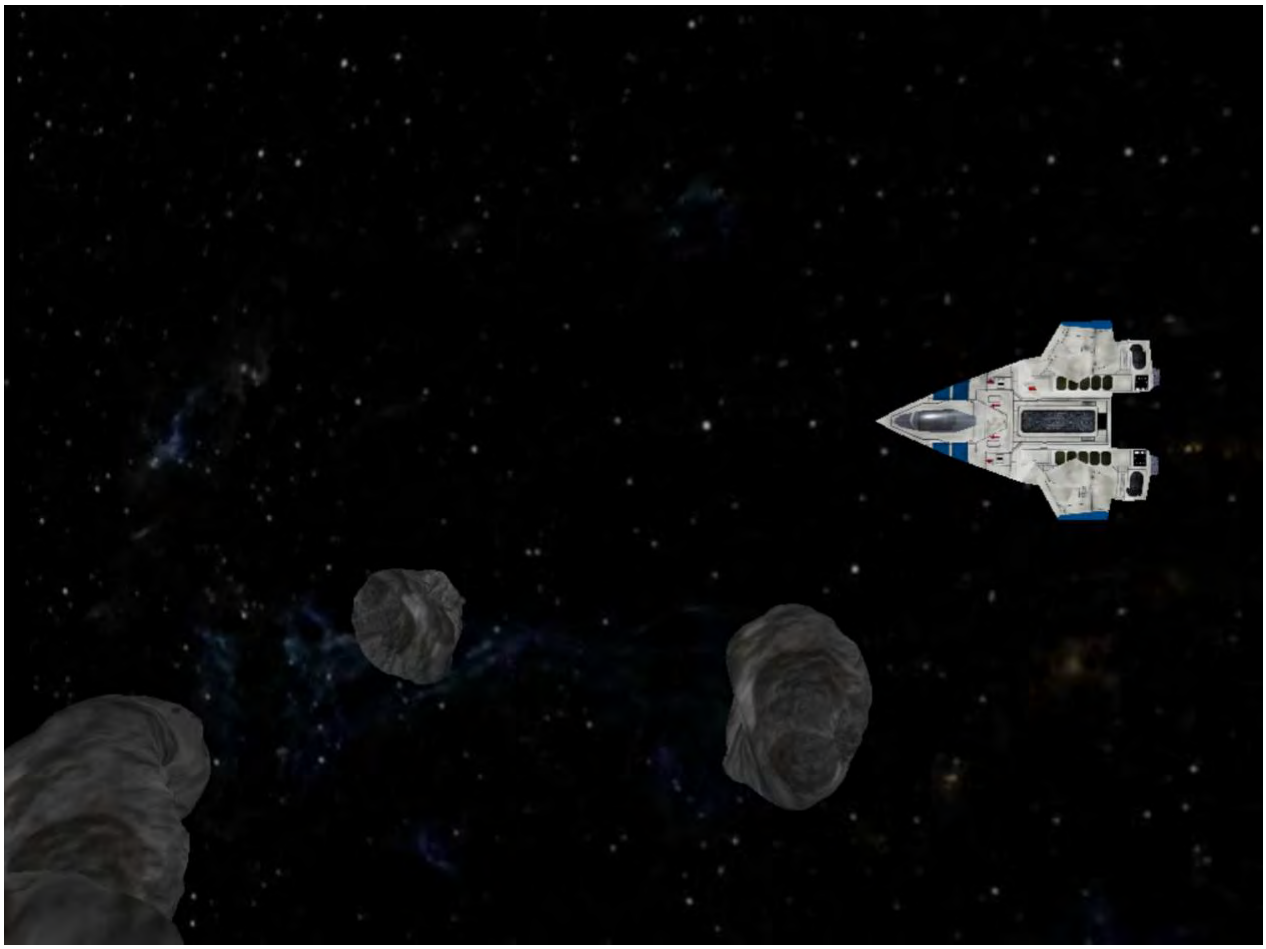
In dieser Aufgabe werden wir Funktionalitäten zum Laden von texturierten Meshes zu unserem Spiel hinzufügen. Dazu werden Ihnen verschiedene Klassen zum Laden von verschiedenen 3D-Formaten und Bitmaps bereitgestellt. Implementieren Sie dazu die Klassen `TriangleMeshFactory` und `TextureFactory`, die jeweils als Singleton realisiert werden sollen. Die Klasse `TextureFactory` soll folgende Funktionalitäten bereitstellen:

- Eine Methode `setBasePath(const string& base)`, die das aktuelle Verzeichnis angibt, in dem nach Bilddateien gesucht wird.
- Eine Methode der Signatur `Texture* TextureFactory::getTexture(const string& filename)`, die ein Texturobjekt zurückgibt. Die Definition der Klasse `Texture` finden sie in der Datei `io/Texture.hpp`. Eine `Texture` besteht aus Pixelwerten in einem `unsigned char` Array, sowie einer Bildbreite- und -höhe. Bestimmen Sie die Endung der übergebenen Datei und laden Sie mit Hilfe der gegebenen Unterklassen der Klasse `io/BitmapReader` die dazugehörigen Daten und erstellen Sie eine passende `Texture`-Instanz als Rückgabewert. Alle Dateien sollen relativ zum durch `setBasePath` gegebenen Verzeichnisgesucht werden.
- Da es Spiel vorkommen wird, dass die selben Texturen mehrfach verwendet werden, soll sichergestellt werden, dass beim Aufruf von `getTexture()` nur dann eine *neue* Textur-Instanz zurückgegeben wird, wenn die entsprechende Datei noch nicht geladen wurde. Für den Fall, dass eine Datei mehrfach geladen wird, sollen sie einen Zeiger auf eine *bereits existierende* Textur zurückgeben. Verwenden Sie dazu in der Klasse eine `std::map` mit passenden Parametern, um von Dateinamen auf Textur-Pointer zu mappen.
- Es sollen -die Dateiformate `.tga`, `.tif`. bzw. `.tiff`, `.ppm` und `.jpg` unterstützt werden. Die entsprechenden Reader-Klassen finden Sie im Ordner `/io/`.
- Nutzen Sie die Factory an den vorgegebenen Stellen (TODOs um Quelltext) in den Dateien `SkyBox.cpp` und `Read3DS.cpp`.

Erstellen Sie analog eine Klasse `TriangleMeshFactory`, die die folgenden Funktionen bereitstellt:

- Eine Methode `setBasePath(const string& base)`, die das aktuelle Verzeichnis angibt, in denen nach 3DModellen gesucht wird.
- Eine Methode mit der Signatur `TriangleMesh* TriangleMeshFactory::getMesh(const string& filename) const`, die je nach übergebener Dateiendung in 3D-Modell lädt. Dazu werden die Klassen `Read3DS` und `ReadPLY` zur Verfügung gestellt, die jeweils Unterklassen von `MeshReader` sind. Auch hier sollen die Dateien im durch `setBasePath` definierten Verzeichnis gesucht werden. Nutzen Sie die dort definierten Methoden, um die Meshes aus den gegebenen Dateien zu instanzieren. Eine Überprüfung auf mehrfaches Laden ist hier nicht erforderlich!
- Nutzen Sie die Klasse in den Dateien `AsteroidField.cpp` und `SpaceCraft.cpp`, um 3D-Modelle zu laden (siehe TODOs).

Wenn Sie die geforderten Funktionalitäten korrekt implementiert haben, sollte das Raumschiff vor einem Sternenfeld umgeben von 3D-Asteroiden gerendert werden:



Aufgabe 3: Range-based For-Loop für die Liste (35 Punkte)

In der Vorgabe für Aufgabe 2 wird eine `std::list` zum Verwalten der Asteroiden-Intanzen in der Klasse `AsteroidField` verwendet. Erweitern Sie die Klasse List in der Datei `util/List.hpp` um die Unterstützung für Range-based For-Loops. Ersetzen Sei die intern verwendete `std::list` durch Ihre Implementierung.

Abgabe:

Checken Sie Ihre Lösung des Aufgabenblattes bis Montag den 17.12.2018, 8:00 Uhr in den Master-Branch des git-Repositorys Ihrer Gruppe ein. Bringen Sie pro Gruppe einen Ausdruck des von Ihnen erstellten Codes mit.