Computergraphik II

Prüfungsvorleistung 02

Szenen Struktur

Abgabetermin: 24.04.2019 11:00 Uhr

Die Aufgaben sind von jedem Teilnehmer eigenständig zu bearbeiten, Gruppenarbeit ist nicht zulässig!
Möglichkeiten zur Abgabe:

- elektronisch über das OPAL-Portal `http://opal.sachsen.de/TUC`
  Eine Einschreibung in die Lerngruppe der Lernressource „571110 Computergraphik II SS 2019“ ist zwingend erforderlich. Falls Sie mehrere Versionen hochladen, beachten wir nur die aktuellste Abgabe. Sollten Sie Fragen haben können Sie sich gerne per Mail (timon.zietlow@informatik.tu-chemnitz.de) an mich wenden.

在上一个练习中，介绍了统一缓冲区对象（UBO）。 与CG2Geometry类一起，您可以使用工具来实现CG2框架的基本结构。

In der vergangenen Übung wurden Uniform Buffer Objects (UBO) eingeführt. Zusammen mit der `CG2Geometry` Klasse besitzen Sie damit das Handwerkszeug die wesentliche Struktur des CG2-Frameworks zu implementieren.[1] In diesem Stand des CG2-Frameworks sind einige Klassen hinzugekommen, welche Sie in dieser PVL vervollständigen, bzw. anpassen sollen. Ziel ist es, die Komponenten, welche zur Organisation der Szene im CG2-Framework verwendet werden zu erstellen und deren Aufgaben und Verhalten zu verstehen. Außerdem sind die hier entwickelten Strukturen relativ flexibel einsetzbar, weshalb Sie diese durchaus auch als Basis für ihre eigenen Projekte verwenden können.

在此版本的CG2框架中，添加了一些您应该在此PVL中完成或调整的类。 目标是创建用于组织CG2框架中的场景的组件，并了解它们的任务和行为。 此外，这里开发的结构相对灵活，这就是为什么你可以将它们作为自己项目的基础。

Konzeptuell soll es eine Klasse geben, die alle Informationen, die zum Rendern der Szene nötig sind, organisiert. Dazu gehören die Kameraparameter, Objektparameter, Geometriedaten und Lichter.

从概念上讲，应该有一个类来组织渲染场景所需的所有信息。 这些包括相机参数，对象参数，几何数据和灯光。

Für jede dieser Informationen ist ein Klasse vorgesehen, die im Folgenden kurz vorgestellt werden. Detailliertere Informationen finden Sie dann in der Aufgabenstellung.

对于这些信息中的每一个，提供了一个类，下面将简要介绍。 然后可以在任务中找到更详细的信息。

`CG2Geometry`(`geometry.h`): Diese Klasse kennen Sie bereits aus der PVL1. Sie kapselt Geometrieinformationen und erlaubt es uns einen Drawcall zu initiieren.

CG2Geometry（geometry.h）：您已经从PVL1中了解了这个类。 它封装了几何信息，并允许我们启动一个drawcall。

`CG2Camera`(`camera.h`): Diese Klasse verwaltet, wie der Name bereits vermuten lässt, die Kameraparameter (Projektions- und View-Matrix).

CG2Camera（camera.h）：顾名思义，这个类管理摄像机参数（投影和视图矩阵）。

`CG2GlobalLights`(`light.h`): Die Verwaltung der Lichter ist etwas komplizierter. Hier wird nicht nur ein Licht verwaltet, sondern ein Array von Lichtern.

CG2GlobalLights（light.h）：灯光的管理有点复杂。 在这里，不仅要管理灯光，还要管理一系列灯光。

`CG2Object`(`object.h`) Das Objekt verwaltet objektspezifische Eigenschaften, wie Position und Orientierung in der Szene (Model-Matrix), und ordnet diese einem entsprechenden Geometrie Objekt zu.

CG2Object（object.h）对象管理特定于对象的属性，例如场景中的位置和方向（模型矩阵），并将它们分配给相应的几何对象。

[1] Es fehlt noch eine Lösung Shader und Materialien zu organisieren, das wird Inhalt der nächsten Übung bzw. PVL sein.
仍然有一个组织着色器和材料的解决方案，这将是下一个练习或PVL的内容。

这些类中的所有对象都应该有效地封装UBO，并允许我们使用适当的方法方便地修改单个值，然后以合理有效的方式将这些更改传输到UBO。

Alle Objekte dieser Klassen sollen effektiv ein UBO kapseln und es uns ermöglichen, komfortabel durch entsprechende Methoden, einzelne Werte zu verändern und diese Veränderungen dann auch halbwegs effizient in das UBO zu übertragen.[2]

CG2Scene类管理上述类的对象。 相机和灯光表示为单独的成员，因为目前仅支持一个相机和一组灯光。 但是，支持几个CG2Object和CG2Geometry对象。 它们存储在名称（std :: string）下，可以使用方法CG2Scene :: getGeometry（const std :: string&name）或CG2Scene :: getObject（const std :: string&name）创建或请求。

Die Klasse `CG2Scene` verwaltet Objekte der oben beschriebenen Klassen. Kamera und Lichter sind dabei als einzelne Member vertreten, da im Moment nur eine Kamera und ein Satz an Lichtern unterstützt wird. Jedoch werden mehrere `CG2Object` und `CG2Geometry`-Objekte unterstützt. Diese werden unter einem Namen (`std::string`) abgelegt und können durch die Methoden `CG2Scene::getGeometry(const std::string& name)` bzw. `CG2Scene::getObject(const std::string& name)` erstellt oder angefordert werden.

Zusätzlich zu diesen Name-Objekt-Paaren speichert die `CG2Scene` die Menge an `CG2Object`s, welche bei einem Aufruf von `CG2Scene::render()` gerendert werden sollen. Wir werden in späteren Einheiten Objekte benötigen, die nicht bei jedem Rendering beachtet werden sollen.

除了这些名称 - 对象对之外，CG2Scene还存储了在调用CG2Scene :: render（）时应该呈现的CG2Object集合。 我们将需要后续单元中的对象，在每次渲染时都不应该考虑这些对象。

Tastaturbelegung:

- Esc : Programm beenden
- F1 / Shift ⇑ + F1 : Wireframedarstellung an/aus
- 1 - 8 : Lichter ein-/ausschalten.
- : Cursor freigeben
- : Cursor einfangen

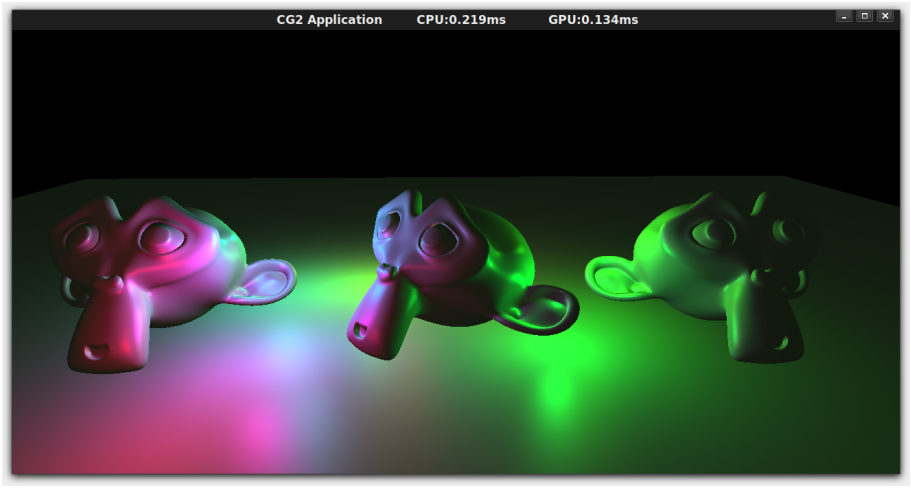

Abbildung 1: Das Ergebnis der PVL2

1. Die CG2Camera Klasse (*3 Punkte*)

Ein Objekt der Klasse `CG2Camera` repräsentiert die Kamera, also eine Projektions- und Viewmatrix, mit der die Szene gerendert werden können soll. Außerdem werden weitere Informationen über die Projektionsmatrix (der near und far-Wert) und die Auflösung des Framebuffers gespeichert. Diese Kamera-Daten sind als

[2] Bei solcher Ähnlichkeit der Klassen bietet sich eine gemeinsame Elternklasse an, von der alle erben. Da diese Struktur den Code aber potentiell verkompliziert haben wir hier darauf verzichtet. (Wer möchte bzw. es sich zutraut kann den Code aber zur Lösung der Aufgaben entsprechend anpassen.)

这种类的相似性提供了一个共同的父类，从中继承所有类。 但是，由于这种结构可能使代码复杂化，我们已经避免这样做了。 （如果您想要或信任自己，可以调整代码来解决任务。）

类CG2Camera的对象表示相机，即投影和视图矩阵，应该使用该投影和视图矩阵来渲染场景。 另外，存储关于投影矩阵（近和远值）和帧缓冲器的分辨率的进一步信息。 这些摄像机数据是作为uboCameraData数据的CG2Camera类的一部分，并且将在UBO中存储在GL侧。 成员变量dirty_ubo指示自上次将数据复制到UBO以来是否已更改某些内容。

`uboCameraData data` ein Teil der `CG2Camera`-Klasse und sollen auf der GL-Seite in einem UBO abgelegt werden. Die Membervariable `dirty_ubo` gibt dabei an, ob sich in `data` etwas geändert hat, seit dem die Daten zuletzt in das UBO kopiert wurden.

他们的任务是确保这些数据存储在相应的UBO中。

Ihre Aufgabe besteht darin dafür zu sorgen, dass diese Daten in einem entsprechenden UBO gespeichert werden.

实现CG2Camera :: createGLObject方法以在ubo_camera中创建适当大小的缓冲区。

- Implementieren Sie die Methode `CG2Camera::createGLObject` so, dass in `ubo_camera` ein Buffer entsprechender Größe angelegt wird.

实现CG2Camera :: upload方法，以便在设置ubo_dirty标志时将当前版本的数据复制到ubo_camera缓冲区。 更新缓冲区后设置变量ubo_dirty = true。

- Implementieren Sie die Methode `CG2Camera::upload` so, dass die aktuelle Version von `data` in den Buffer `ubo_camera` kopiert wird, wenn das flag `ubo_dirty` gesetzt ist. Setzen Sie die Variable `ubo_dirty = true` nachdem Sie den Buffer aktualisiert haben.

CG2Camera :: use（）方法应确保UBO是最新的，然后将其绑定到GL_UNIFORM_BUFFER的相应子表。 有关索引列表，请参阅bindings_locations.h

- Die Methode `CG2Camera::use()` soll sicherstellen, dass das UBO aktuell ist, und es dann an das entsprechende Subtarget von `GL_UNIFORM_BUFFER` binden. Eine Liste der Indices finden Sie in `bindings_locations.h`

Anpassen der Shader

Vertex Shader目前使用直接设置的“ normalen ”制服。 调整两个着色器（data/shaders/example.*.glsl），以便使用来自UBO的数据而不是正常的制服。 已经准备好相应的Uniform块。

Der Vertexshader verwendet aktuell noch die direkt gesetzten 'normalen' Uniforms. Passen Sie die beiden Shader (`data/shaders/example.*.glsl`) so an, dass anstelle der normalen Uniforms die Daten aus dem UBO genutzt werden. Die entsprechenden Uniform Blöcke sind schon vorbereitet.

如果到目前为止您已经解决了任务，您应该能够在场景中自由移动相机。 分析camera.cpp中的代码，该代码转换相机控件并尝试理解它。

Wenn Sie die Aufgabe bis hier richtig gelöst haben sollten Sie in der Lage sein die Kamera frei in der Szene zu bewegen. Analysieren Sie den Code in `camera.cpp`, der die Kamerasteuerung umsetzt und versuchen Sie diesen nachzuvollziehen.

2. Die CG2Object Klasse (*1 Punkt*)

类CG2Object的对象表示所有“ 每个对象 ”属性。 这些包括对象的几何（CG2Geometry）及其在空间中的位置。 实现类似于CG2Camera类的类CG2对象的方法

Objekte der Klasse `CG2Object` repräsentieren alle 'pro Objekt' Eigenschaften. Dazu gehören die Geometrie des Objekts (`CG2Geometry`) und dessen Lage im Raum.[3] Implementieren Sie die Methoden der Klasse `CG2Objekt` analog zur Klasse `CG2Camera`

- Das `CG2Object` hat keine `use()`-Methode, sondern die Methode `CG2Object::render()`. Hier wird aktuell die `render`-Methode der Geometrie aufgerufen. Die Synchronisation und das Binden des UBOs soll davor passieren.

CG2Object没有use（）方法，但方法CG2Object :: render（）。 这里调用几何的当前方法。 UBO的同步和绑定应该在此之前发生。

Anpassen der Shader

Passen Sie den Vertex-Shader so an, dass die Informationen aus dem UBO verwendet werden um die Vertexdaten zu transformieren. Auch hier ist der entsprechende Interface Block bereits vorgegeben. Wenn Sie alles richtig gelöst haben, sollte sich der Affenkopf drehen.

调整顶点着色器以使用UBO中的信息来变换顶点数据。 同样，已经指定了相应的接口块。 如果你已经正确地解决了所有问题，那么猴子头应该转动。

3. Mehrere Objekte (*1 Punkt*)

Passen Sie die Methode `CG2App::init_gl_state()` so an, dass drei Affen nebeneinander gezeichnet werden. Achten Sie darauf, dass dafür nur **ein** `CG2Geometry` Objekt angelegt wird.

调整CG2App :: init_gl_state（）方法并排绘制三只猴子。 确保只为它创建一个CG2Geometry对象。

[3] Später kommt noch das verwendete Material dazu.
后来使用的材料。