Fakultät Informatik/Mathematik

Lehrstuhl für Computergrafik

Hochschule für
Technik und Wirtschaft
Dresden
University of Applied Sciences

BELEUCHTUNG UND RENDERINGI

BELEGARBEIT

Dokumentation von

Daniel Kegel (), Eric Schmidtgen (46562) und Linda Garten (44371)

Betreuender Hochschullehrer

Prof. Dr. Marco Block-Berlitz

Eingereicht am: 19. Januar 2020

INHALTSVERZEICHNIS

1 EINLEITUNG UND AUFGABENSTELLUNG

In Gruppen von bis zu drei Personen dürfen kleine Projektideen von der Konzeption bis zur Umsetzung durchgeführt werden. Die Inhalte müssen allerdings mit den Dozenten vorab abgesprochen werden. Grundvoraussetzung für die Projektideen sind Innovation und Interaktion. Es dürfen auch Game-Engines verwendet werden. Bei der Bewertung wird der Eigenanteil entsprechend berücksichtigt. Teil 2 fließt mit mindestens 50kompensieren. In jedem Fall ist die Rücksprache mit den Dozenten erforderlich.
Das Thema für die Projekte lautet in diesem Semester: SCHATTEN

Die Abgabe der Arbeit erfolgt im letzten Praktikum des Wintersemesters 2019/20.

2 VORSTELLUNG DER LÖSUNG

2.1 IDEE

Unsere Idee für die Umsetzung der Aufgabe ist es, dass wir in UNITY eine Taschenlampe durch eine Welt steuern können. Wir sollen dabei die Taschenlampe spielen und durch ein Haus "laufen"pro können. Unity bietet den Vorteil, dass einfache Objekte wie Lichtquellen oder grafische Primitive (Ebenen, Würfel, Kugeln) einfach im Editor erzeugt werden können. Somit sparen wir uns die Berechnung des Lichtes beziehungsweise des Schattens und wir erzielen schnell Erfolge.

2.2 UNITY ASSET STORE

Damit wir nicht viel Zeit beim modellieren verschwenden, werden wir auf bereits vorgefertigte ASSETS zurückgreifen. Für die Taschenlampe verwenden wir das freie Asset von RRFREELANCE[1].

2.3 ERSTER ÜBERBLICK

Nach ein paar Stunden im Probieren mit Unity (Rigidbody, Gravity, Animator, etc.) laden wir unser Asset (siehe Abb. 1). Wir probieren erst einmal die schon fertigen Komponenten des Assetes und stellen schnell fest, dass wir mit der Physik nicht zufrieden sind. Wir bemerken, dass die Physik der Kamera in Kombination mit dem Licht nicht stimmig ist (siehe Abb. 2). Dies werden wir mit weiteren Klassen anpassen, da eine Taschenlampe und Kamera nur mit einem anderen Körper gut harmonieren.

2.4 KLASSEN

2.4.1 PLAYER

Unser Spiel selbst ist ein kleines TIC-TAC (Kapsel), was von Unity bereitgestellt wird. In diese Gruppe haben wir die Taschenlampe und die Kamera gepackt. Unity selbst fasst damit eine Gruppe zusammen. Nun müssen wir nur noch sagen, wie schnell sich unser Spieler (Player) bewegen und drehen soll. Dafür legen wir uns zwei FLOAT-Variablen an und initialisieren mit getesteten Werten:

```
1 public float movementSpeed = 10;
2 public float turningSpeed = 60;
3
4 void Update() {
5   float horizontal = Input.GetAxis("Horizontal") * turningSpeed * (
      Time.deltaTime * 4);
6   transform.Rotate(0, horizontal, 0);
```

[1] RRFreelance, "Flashlight", im Unity Asset Store, https://assetstore.unity.com/packages/3d/props/electronics/flashlight-18972, abgerufen am: 14.01.2020

```
7
8   float vertical = Input.GetAxis("Vertical") * movementSpeed * Time
        .deltaTime;
9   transform.Translate(0, 0, vertical);
10 }
```

Die Werte für die jeweilige Achse lesen wir aus, multiplizieren unsere erstellten Variablen dazu und die Zeit als Faktor der Geschwindigkeit. Da wir dies in der UPDATE()-Funktion machen, können wir uns in Echtzeit bewegen.

2.4.2 FLASHLIGHT

Um die Physik der Taschenlampe anzupassen, müssen wir ein Script anlegen und etwas C# programmieren.
Am Anfang legen wir ein paar Variablen an und initialisieren diese direkt mit Werten. Wir benötigen diese später für unsere Physik.

```
1  public class FlashLight : MonoBehaviour {
2    public float horizontalSpeed = 2f;
3    public float verticalSpeed = 2f;
4    float mainSpeed = 100.0f; //regular speed
5    float shiftAdd = 250.0f; //multiplied by how long shift is held.
         Basically running
6    float maxShift = 1000.0f; //Maximum speed when holdin gshift
7    float camSens = 0.25f; //How sensitive it with mouse
8    private Vector3 lastMouse = new Vector3(255, 255, 255); //kind of
         in the middle of the screen, rather than at the top (play)
9    private float totalRun = 1.0f;
10   ... }
```

In der UPDATE() - Funktion lesen wir die X und Y-Werte der Mausposition aus und geben diese an die Transformation des Elementes weiter.

```
1  void Update() {
2      float h = horizontalSpeed * Input.GetAxis("Mouse X");
3    float v = verticalSpeed * Input.GetAxis("Mouse Y");
4    this.transform.Rotate(v, 0, h);
5    ...
6  }
```

Damit wir uns nun noch in der Welt mit unserer Taschenlampe bewegen können, implementieren wir uns noch ein paar Tastenkombinationen. Dafür arbeiten wir mit Vector3D.

```
1  private Vector3 GetBaseInput() {
2    Vector3 p_Velocity = new Vector3();
3    if (Input.GetKey(KeyCode.W)) {
4      p_Velocity += new Vector3(0,0.03f,0);
5    }
6    if (Input.GetKey(KeyCode.S)) {
7      p_Velocity += new Vector3(0, -0.03f,0);
8    }
9    if (Input.GetKey(KeyCode.A)) {
```

```
    p_Velocity += new Vector3(0.03f, 0, 0);
  }
  if (Input.GetKey(KeyCode.D)) {
    p_Velocity += new Vector3(-0.03f, 0, 0);
  }
  if (Input.GetKey(KeyCode.Y)) {
    p_Velocity += new Vector3(0, 0, -0.03f);
  }
  if (Input.GetKey(KeyCode.X)) {
    p_Velocity += new Vector3(0, 0, 0.03f);
  }
  return p_Velocity;
}
```

2.4.3 RESET

Da unsere Welt, insbesondere unser Haus, etwas fehlerbehaftet ist und wir oft mit unserem Spieler in Wänden stecken bleiben oder wir GLITCHEN, haben wir uns dafür entschieden einen Reset-Button einzufügen. Beim Klicken der Taste R wird der Spieler an die Position (0,1,0) zurückgesetzt. Die haben wir mit einem 3D-Vektor abgebildet:

```
if (Input.GetKeyDown(KeyCode.R)) {
  transform.position = new Vector3(0, 1, 0);
 }
```

2.4.4 MOUSELOOK

Die Klasse MOUSELOOK rotiert und transformiert auf Basis unseres Maus-Deltas. Ein Minimum und Maximum begrenzt unsere Rotation. Das erstellt Script wird der Kamera zugewiesen, welche ein Kind des TICTAC'S ist.

2.4.5 PLAYERFOLLOW

Damit der Spieler sich so natürlich wie möglich bewegt, brauchen wir noch einen gewissen SMOOTH-Faktor. Dieser beeinflusst die Bewegung deutlich. Die Richtung des zurückgegeben Vektors wird durch den Winkel interpoliert und seine Größe wird zwischen den Größen von aktueller Position und neuer Position interpoliert.

```
void Update () {
  Vector3 newPos = PlayerTransform.position + _cameraOffset;
  transform.position = Vector3.Slerp(transform.position, newPos,
     SmoothFactor);
  if (LookAtPlayer)
    transform.LookAt(PlayerTransform);
}
```

2.4.6 TASTATURBELEGUNG

W	Bewegung nach vorn
A	Bewegung nach links
S	Bewegung nach hinten
D	Bewegung nach rechts
Y	Bewegung nach unten
X	Bewegung nach oben
R	Zurücksetzen
F	Licht aus bzw. anschalten
G	Lichtfarbe wechseln
LEFT SHIFT	Sprint

2.5 WELT - DESERTSETHOUSE

Damit wir mit unserer Taschenlampe nicht einfach auf einer einfachen und langweiligen Oberfläche laufen, werden wir ein altes Wüstengebäude nutzen. Auch dies haben wir im Unity AssetStore gefunden[2]. Damit es für unser Projekt passt, mussten wir noch einige Anpassungen in Maya vornehmen. So haben wir die Türen geöffnet, damit wir sauber mit der Taschenlampe durchlaufen können (siehe Abb. 3). Zudem wurden noch einige Wände vervollständigt (siehe Abb. 4).

2.6 PROBLEME BEIM IMPORT

Beim Import des Maya-Files in Unity gab es Probleme mit dem Schatten und den Texturen. So wurden die Wände im Haus nicht gezeichnet beziehungsweise ausgeleuchtet. Nach einer kurzen Recherche haben wir den Fehler gefunden und konnten mit Hilfe von DOUBLE-SIDED-SHADERS das Problem fixen[3].

2.7 ERGEBNIS

Mit dem abschließenden Ergebnis sind wir sehr zufrieden. Wir alle haben das erste mal mit Unity gearbeitet und waren überrascht, wie schnell man Fortschritte macht und zu einem Ergebnis kommt (siehe Abb. 5). Die Möglichkeiten schienen schier unbegrenzt.
Wir haben einen Spieler, den man definierten Tasten bewegen kann. Umschauen kann man sich mit der Maus. So ist es möglich durch das Haus zu navigieren (siehe Abb. 6 und 7).

[2] Crazy Cool, "Desert Set House", im Unity Asset Store, https://assetstore.unity.com/packages/3d/environments/historic/desert-set-house-138105, abgerufen am: 14.01.2020

[3] Ciconia Studio, "Double Sided Shaders", im Unity Asset Store, https://assetstore.unity.com/packages/vfx/shaders/double-sided-shaders-23087, abgerufen am: 14.01.2020

3 ABBILDUNGSVERZEICHNIS

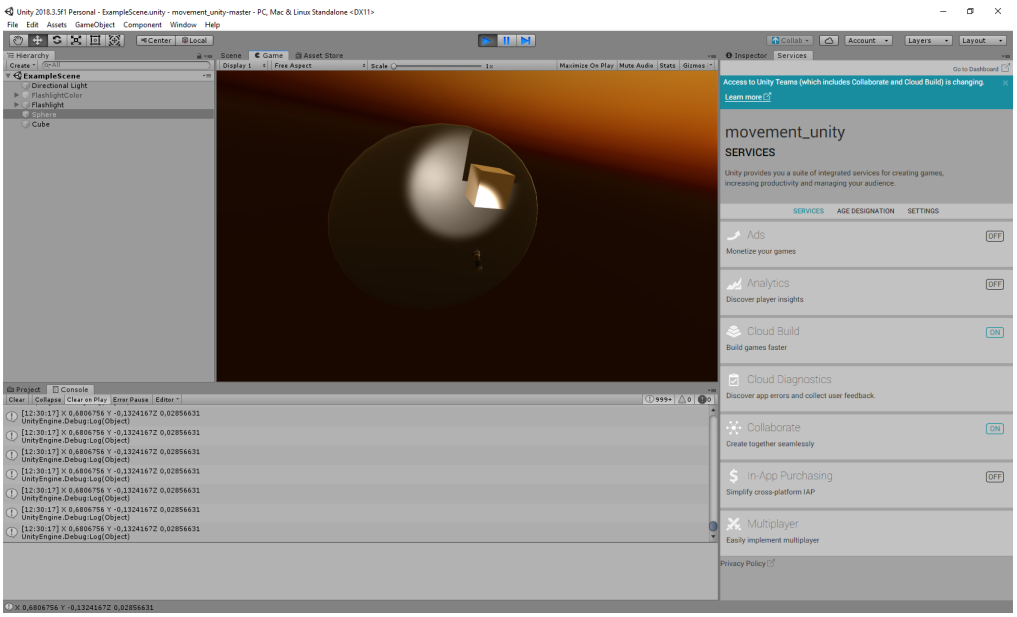

Abbildung 1: Unity UI - Flashlight Szene mit Debug Ausgabe im Modus: Run

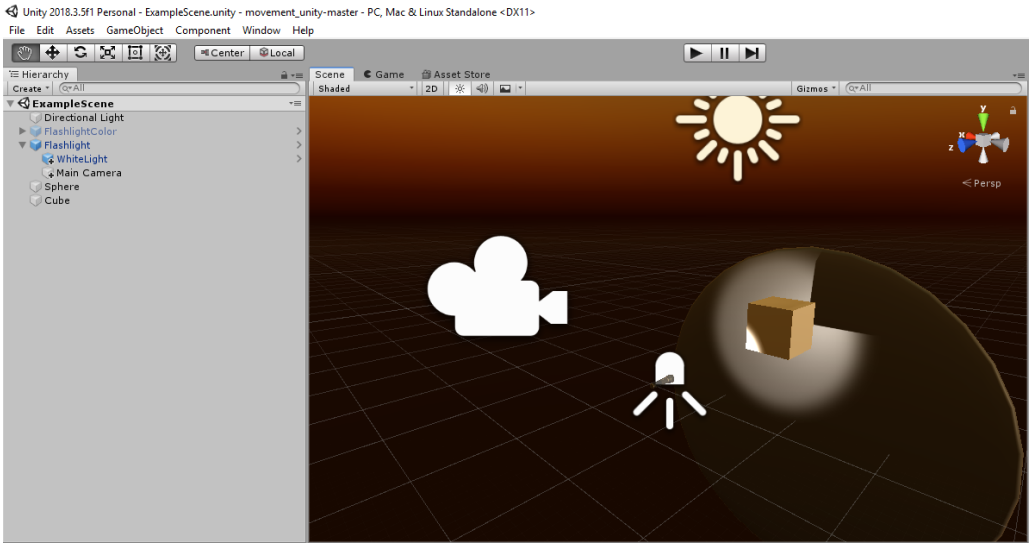

Abbildung 2: Unity UI - Flashlight Szene im Build-Modus

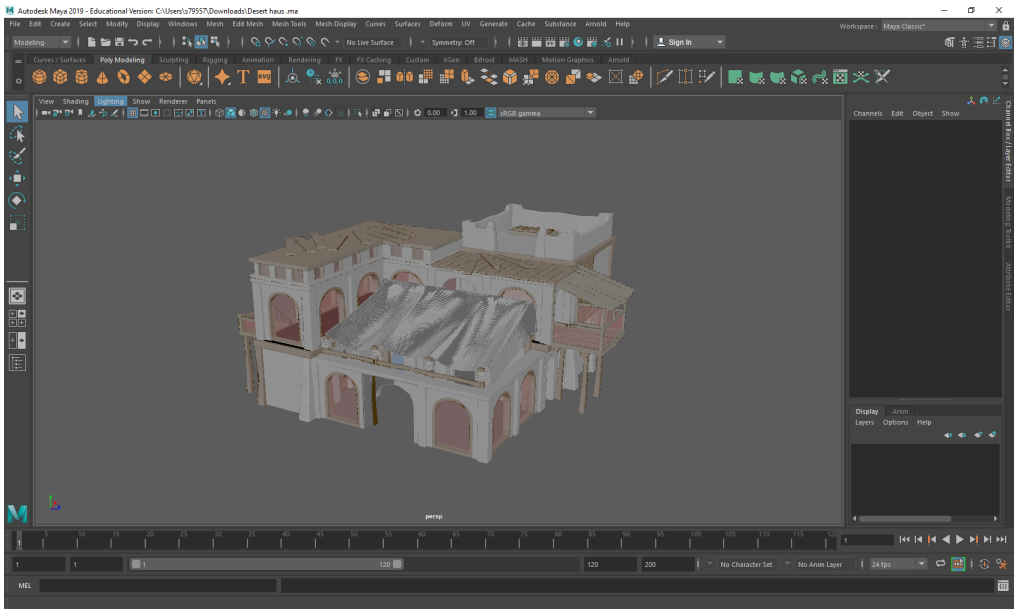

Abbildung 3: Autodesk Maya - Sicht von vorn

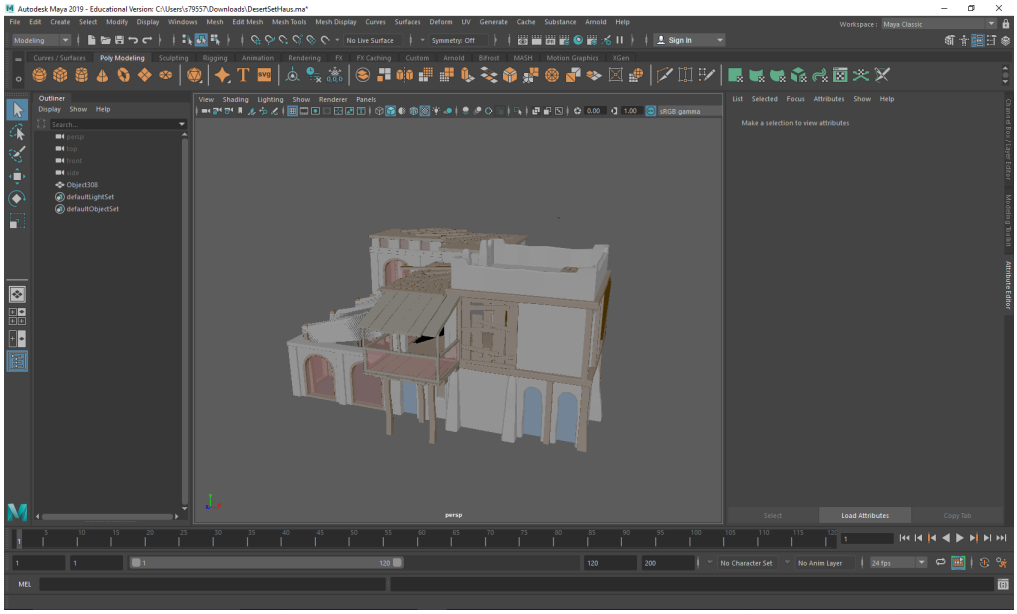

Abbildung 4: Autodesk Maya - Sicht von der Seite

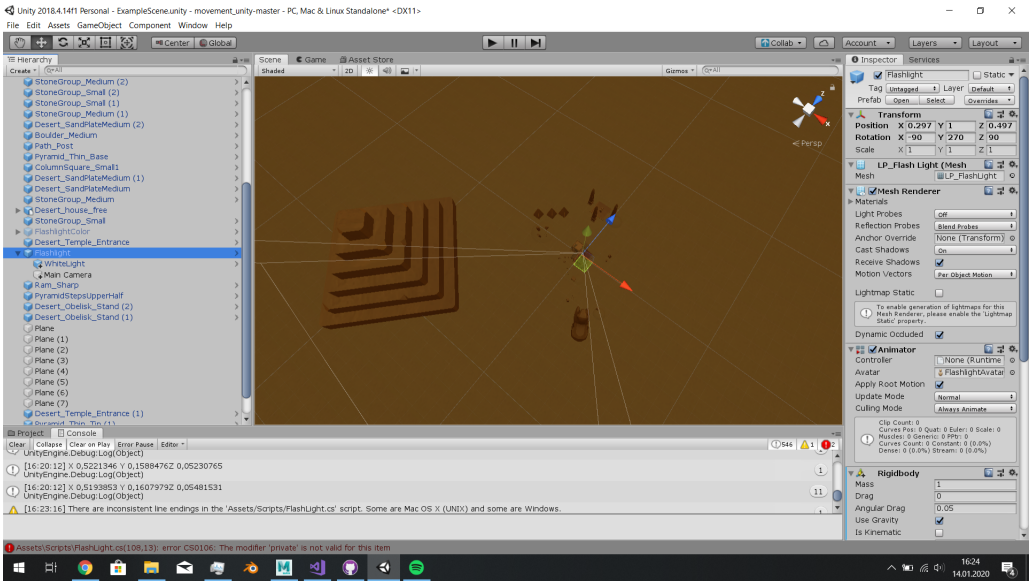

Abbildung 5: Unity UI - Überblick über die Welt; links der Outliner des Projektes

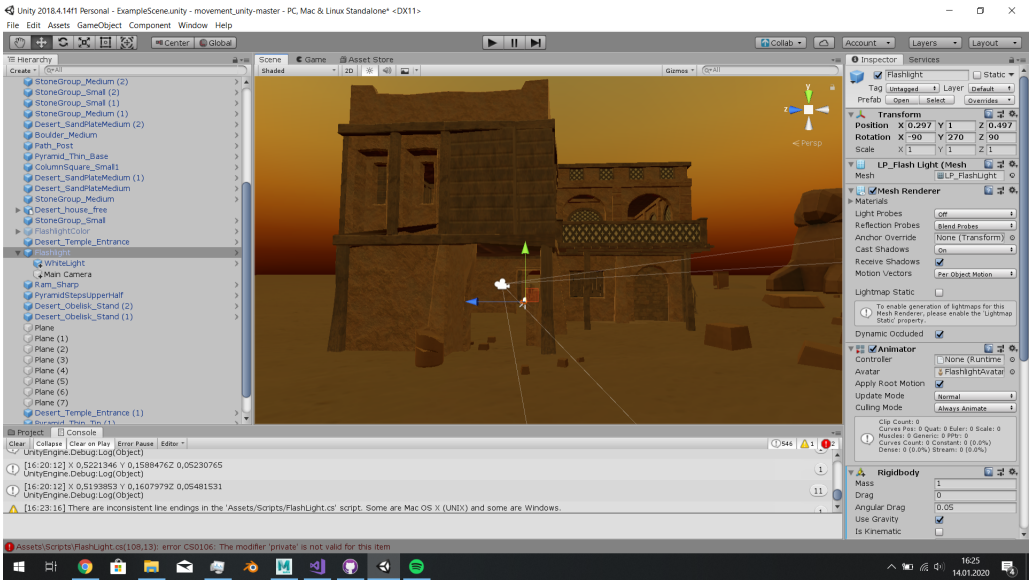

Abbildung 6: Unity UI - Blick auf das Desert House

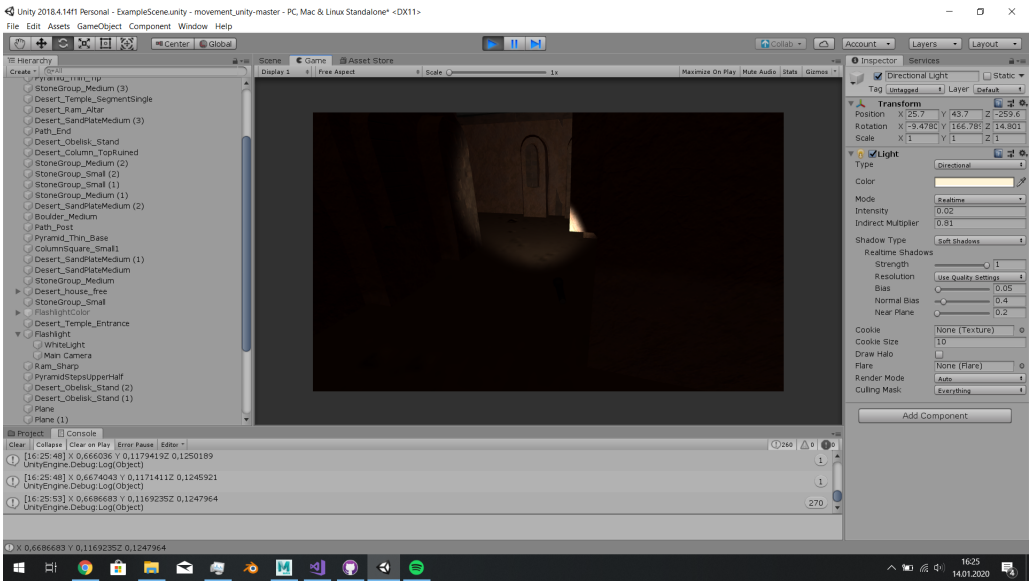

Abbildung 7: Unity UI - Projekt im Run Modus