Aufgabe 5: Simulations & Cloth

(Neue) Techniken:

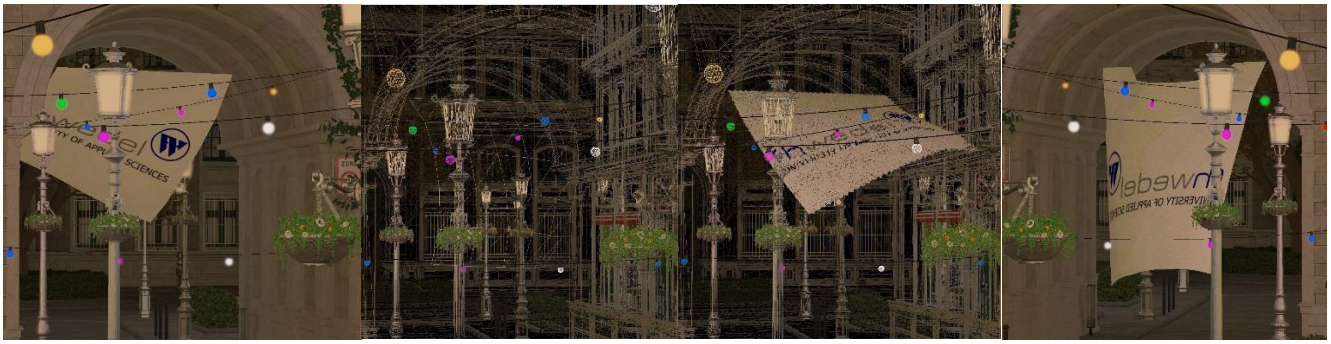
Compute Shader, Simulationen, Stoffsimulationen, Prozedurale Texturen

Beschreibung:

Im Mittelpunkt dieser Aufgabe steht die Simulation, für die der Compute Shader (GPGPU Programmierung) genutzt werden soll.

Der Umgang mit dem Compute Shader könnt ihr den Präsentationsfolien entnehmen sowie, dem im Samples Repo bereitgestellten GPU Compute Shader Sample oder dem Blur Sample.

Für die Bearbeitung der Aufgabe muss OpenGL 4.3 und GLSL 4.30 (oder jeweils höher) verwendet werden.



Kernanforderung:

- Nutzt Indices für das Mesh der Cloth Simulation. Legt diese in ein Buffer-Objekt und rendert mit dem Draw-Command glDrawElements.
- Implementiert die Simulation unter Nutzung des Compute Shaders und arbeitet direkt auf dem Mesh. Achtet dabei darauf, Ergebnisse zu erzeugen, die unabhängig von der Ausführungsreihenfolge der Compute Shader Invocations sind.
- Berechnet nach der Simulation neue Normalen und Tangenten für die Vertices.

Weitere Funktionalitätsanforderungen:

- Die Simulation wird in 2 Copmute Shadern berechnet.
    - Zuerst wird mit der *Iterativen Constraint Satisfaction Methode* parallel über alle Constraints iteriert
    - In einem 2. Durchlauf wird mittels der *Verlet Integration* parallel über die Massen (-partikel) des Meshes iteriert
- Nutzt für die Windkraft eine Noise Funktion die sich Orts- & Zeitabhängig (World Position) ändert
- Die Animation soll pausiert werden können.
- Nutzt weiterhin Tessellation um das Mesh abhängig von der Distanz zu verfeinern.
- Die Auflösung des Mesh lässt sich mittels Tastendruckes verändern. Achtet dabei darauf, dass die Simulationsbuffer mit angepasst werden.
- Weiterhin soll die Lichtberechnung mittels Deferred Rendering funktionieren, das Mesh soll dementsprechend auf den Gbuffer gerendert werden
- Die Berechnung der Normalen und Tangenten soll in einem zweiten Compute Shader parallel durchgeführt werden, achtet hierbei auf effiziente Berechnung und dass Additionen ggf. Atomar behandelt werden.
- Verteilt das Clothmesh an verschiedenen Orten in der Stadt Szene (achtet darauf, dass dies bei dem Deferred Rendering sowie weiteren Rendertechniken korrekt beachtet wird). [solltet ihr auf diesen Punkt der Bewertung verzichten wollen ist es okay, wenn ihr die gesamte Simulation & Rendering in einem neuen Programm durchführt, anstelle eurer Lösung aus Aufgabe 1-4 zu verwenden]

Tastatur-Belegung (Vorschlag):

Taste	Funktion
Esc	Beenden des Programms
F1	Toggle Fullscreen
F2 [Optional]	Toggle Tangetspace
F3 [Optional]	Toggle Wireframe
F10	Reload Shader
1	Toggle Normalmapping
2	Toggle Parallaxmapping
3	Toggle Tessellation
4	Toggle SSAO

H/h	Toggle Hilfe
F/f	Goggle FPS
G/g	Toggle Grid/Szene
D/d	Toggle Directional Light Quelle
L/l	Anzeige der Lichtquellen
N/n	Grid-Auflösung erhöhen
M/m	Grid-Auflösung verringern
Q/q	Gamma-Wert erhöhen
W/w	Gamma-Wert verringern
E/e	Exposure erhöhen
R/r	Exposure verringern
J/j	AO Radius erhöhen
K/k	AO Radius verringern
B/b	Toggle SSAO Blur
<TODO>	<TODO>
U/u	Der Versatz der Vertices entlang der Normale wird erhöht.
I/i	Der Versatz der Vertices entlang der Normale wird verringert.
P/p	Zeitabhängige Animation pausieren
S/s	Umschalten zwischen (Debug) Kachelansicht oder Finaler Szene
STRG + LMB	Kamera Rotation
STRG + MMB	Pan Kamera
STRG + RMB (ggf. Maus Wheel)	Zoom Kamera (ggf. Versetzung entlang Orientierung)

Shader (Vorschlag):

Shader (Komponenten)	Beschreibung
SSAOShader *(Vertex & Fragment)*	Berechnung der Screen Space Ambienten Verdeckung
GBufferShader *(Vertex & TessellationControl & TessellationEvaluate & Fragment)* (Aufg 3)	Rendert das Modell bzw. das Grid (die Geometrie & Material Informationen) in den G-Buffer (MRT). Ggf. mit Tessellation & Displacement. *
DirLightShader *(Vertex & Fragment)*	Directional Light-Beleuchtung *(Mit SSAO Abschwächung)*

PointLightShader *(Vertex & Fragment)*	Point Light-Beleuchtung *(Mit SSAO Abschwächung)*
NullShader *(Vertex & Fragment)* (Aufg 3)	Für den Stencil Test
SkyShader *(Vertex & Fragment)* (Aufg 1)	Zeichnet die Skymap im Hintergrund
BlurShader *(Vertex & Fragment)* (Aufg 3)	Shader, für den Blureffekt (mit advanced 2-Pass Filter; es reicht ein Shader mit einer entsprechenden Abzweigung)
PostProcessShader *(Vertex & Fragment)* (Aufg 3)	Shader, dient zum Zusammenführen der Ausgabe-Textur mit der Textur der Punktlichtquellen, die mittels Blur (mehrfach) gefiltert wurde sowie der Gamma-Korrektur und dem Tone-Mapping
TangentShader *(Vertex & Geometrie & Fragment)* (Aufg 2) *[optional]*	Zeichnet die 3 Achsen des Tangent Spaces, es ist darauf zu beachten, dieselbe Berechnung wie beim ModelShader zu nutzen.
TextShader *(Vertex & Fragment)* (Aufg 1)	Kann Text auf dem Display ausgeben
lightVisShader *(Vertex & Fragment)* (Aufg 3) *[optional]*	Zeichnet die Punkt-Lichtquelle als Primitive Geometrie (Cube oder Sphere)
<TODO>	<TODO>
<TODO>	<TODO>

* Hier findet das Normalmapping und Parallaxmapping statt.

Tipps:

1) Legt für den Anfang nur 4D-Vektoren in die Shader Storage Buffer, um Padding-Probleme (siehe Folien der Aufgabenvorstellung) zu verhindern.
2) Testet vor der Simulations Implementation aus, ob der Compute Shader richtig implementiert wurde. Z.B. durch konstantes setzten der Y-Koordinate bzw. abhängig von der
3) Achtung bisher können lediglich Nvidia Grafikkarten atomare Floating Point Funktionen in O-penGL Compute Shader durchführen.
   Daher müssen atomare Operationen mit int bzw. uint durchgeführt werden.
   Die GLSL Funktionen floatBitsToUint und uintBitsToFloat können behilflich sein.
   Die GLSL Operation atomicCompSwap kann verwendet werden um eine atomare „vergleich-tausch“ Operation durchzuführen.
   a. Beispiel siehe Folie
4) Wenn die Simulation in die Scene eingebaut wird, ist es okay, wenn ihr diese nicht zusätzlich auf dem Standard Grid durchführt, damit ihr Dimensionskomplikationen vermeidet.

Weitere (freiwillige) Funktionalitäten:

1) **[EXTRA PUNKT]** Es soll mittels prozeduraler Texturen eine Strickmuster Textur erstellt werden. (siehe Folien)

2) Anstelle der OpenGL Compute Shader kann auch ein Cuda oder OpenCL Kernel geschrieben und genutzt werden. (Generell ist es ratsam im simplen OpenGL Context (wie dieser Aufgabe) den Compute Shader zu nutzen um Overhead durch OpenCL oder Cuda zu vermeiden)
3) Wendet die Simulation auf andere Meshes z.B. Klamotten (Tshirt etc) an. Achtung hier müsst ihr euch überlegen, wie ihr die Federn/(Constraints) an den Compute Shader übergebt.