Aufgabe 3: Light it up!

(Neue) Techniken:

Deferred Rendering, Multi-Render Targets, G-Buffer, Framebuffer, Post Processing, Blooming

Beschreibung:

In dieser Aufgabe soll Deferred Shading implementiert werden. Ebenfalls behandelt die Aufgabe verschiedene, in der Vorlesung vorgestellte Post-Processing Effekte. Im Gegensatz zu dem bisher verwendeten Forward-Rendering werden in einem ersten Renderdurchlauf Geometrie-Daten je nach Tiefe & Clipping mittels Multi Render Target Technik in den G-Buffer gerendert. Im 2. Schritt können wir mittels Lightvolumen mit einer größeren Anzahl an Lichtquellen unsere Szene beleuchten, auf Grundlage der bereits vorgerenderten Geometrie-Daten. Ebenfalls nutzen wir HDR, um ein besser ausbalanciertes Bild zu bekommen und mittels Tone Mapping wird dies am Ende in einen entsprechenden Farbbereich gebracht.

Es soll die in Aufgabe 1 & 2 angefertigte Lösung weiterverwendet werden.

Für die Bearbeitung dieser Aufgabe sollte OpenGL 4.0 und GLSL 4.0 (oder höher) verwendet werden.

Kernanforderung:

- Mindestens ein Framebuffer Object wird erstellt und die Attachments mit Texturen versehen.
- Mehrere G-Buffer Layer/Textur-Attachments (Position, Normale...) werden unter Verwendung eines Framebuffer Objects auf unterschiedliche Texturen geschrieben.

- Die G-Buffer-(Geometrie-)Layer/Textur-Attachments werden zu einem Bild kombiniert.

Weitere Funktionalitätsanforderungen:

- Es soll (min.) die Texturattachments: Position, Normal, AlbedoSpec, Emission & Final geben. Der Tiefenbuffer soll in dieser Aufgabe explizit ein Render-Buffer-Objekt sein, da dies in dieser Aufgabe nicht gesampelt werden muss.
- Die im Geometry-Render-Pass in den G-Buffer gerenderten "**deferred Textures**" (Albedo, Normal, Position, Emission) können auf dem Screen (nebeneinander/untereinander) angezeigt werden. Der 4. verbleibende Platz kann schwarz bleiben, kann die fertige Szene anzeigen, kann die gefiltere Emission Textur s.u. oder kann den spekularen Anteil anzeigen (letzteres wird etwas schwerer, wenn der spekulare Anteil in dem Alpha-Channel der Albedo-Spec Textur gespeichert ist).
- Die in A1 und A2 geforderten Shading Stages sind nicht mehr gefordert, anstelle dessen soll zwischen dem finalen Ausgabebild und der oben beschriebenen Kachelansicht umgeschaltet werden.
- Es gibt ein **Directional Light** (Grundlicht), hierbei wird die Szene auf ein Quad gerendert mit der Beleuchtungsberechnung abhängig von dem Directional Light. (Ein Directional Light besitzt keine "Bounding" Körper, da es infinitiv ist). Diese ist per Tastendruck an-/ausschaltbar.
- Für die Lichtberechnung werden **Light-Volumes** verwendet, um den nicht sichtbaren Overhead möglichst gering zu halten. Für die **Punktlichtquellen** wird in der Szene jeweils eine Sphere (Radius abhängig von Lichteigenschaften) mit Licht Berechnung dieser Lichtquelle gerendert. Hierbei ist darauf zu achten, dass der Depth- & Stencil-Test (Stencil Pre-Rendering!) korrekt ist. Die Ergebnisbilder der einzelnen Renderdurchläufe werden mittels Blending übereinandergelegt. Die Lichtquellen müssen nicht mehr visualisiert werden, da die Visualisierung aus dem Emissionbuffer resultiert.
- Die Punktlichtquellen-Attribute sollen aus der bereitgestellten **JSON** Datei eingelesen werden, diese enthält die Positionen passend zu den Geometrien in der Szene.
- Es wird eine **korrekte Blinn-Phong** Beleuchtung für die Punktlichtquelle sowie für die Richtungslichtquelle implementiert (aus Effizenzgründen & Lesbarkeit jeweils in einem eigenen Shader).
- Die Szene wird in (fast) allen Rendervorgängen als **HDR Framebuffer(n)** [16 Bit & Gleitkommaformat] gerendert, erst im letzten Schritt vor der Ausgabe auf den Bildschirm, wird mittels **Tone Mapping (Exposure Tone Mapping)** der HDR in den Low Dynamic Range-Farbraum transformiert, hierbei soll der Exposure-Wert per Tastatur steuerbar sein. (ACHTUNG: *Texturen müssen beim Laden in den linear Space transformiert werden!*)
- Es findet eine **Gamma-Korrektur** statt, der Gamma-Wert liegt per Default bei 2.2, kann aber verändert werden des weiteren soll der Exposure-Wert auch veränderbar sein.
- Mittels **Bloom** sollen die Lichtquellen (EmissionBuffer) zum "Leuchten"/"Glühen" gebracht werden. Hierbei wird der Blur-Shader-mehrfach (PingPong-Buffer) gefiltert um später mit dem Gerenderten Bild zusammengefügt zu werden. Das Anzeigen der Lichtquellen incl. Blooming soll per Tastatur umschaltbar sein. Es soll explizit ein **two-pass Gaussian Blur** verwendet werden.
- Die Framebuffer und Texture-Attachments sollen auch bei veränderter Fenstergröße funktionieren.

Tastatur-Belegung (Vorschlag):

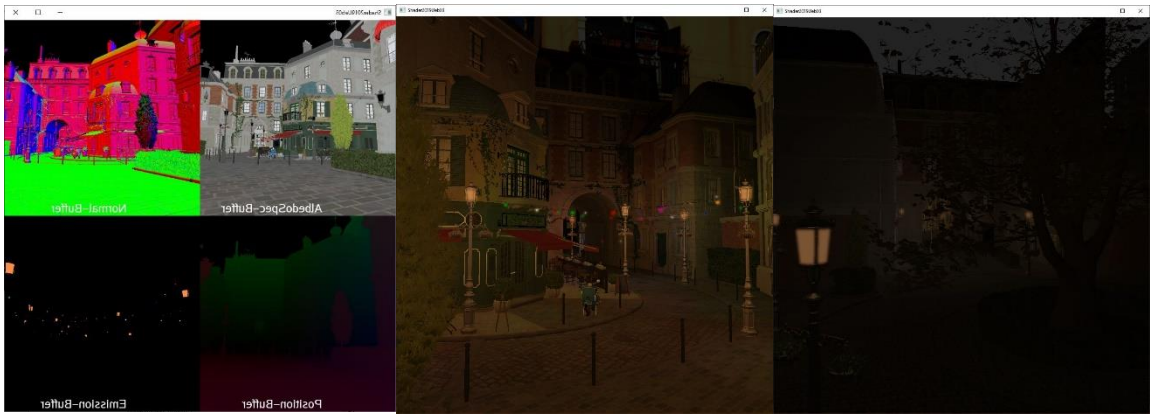
Taste	Funktion
Esc	Beenden des Programms
F1	Toggle Fullscreen
F2 [Optional]	Toggle Tangetspace
F3 [Optional]	Toggle Wireframe
F10	Reload Shader
1	Toggle Normalmapping
2	Toggle Prallaxmapping
3	Toggle Tessellation
H/h	Toggle Hilfe
F/f	Goggle FPS
G/g	Toggle Grid/Szene
D/d	Toggle Directional Light Quelle
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
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)

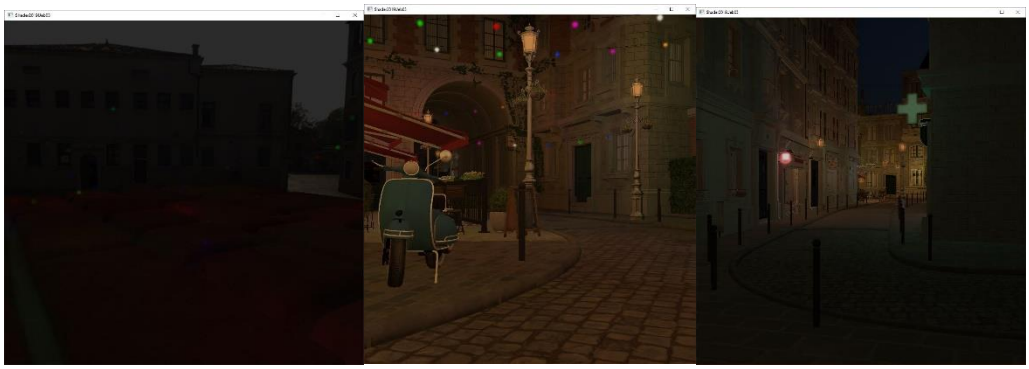
Shader (Vorschlag):

Shader (Komponenten)	Beschreibung

GBufferShader *(Vertex & TessellationControl & TessellationEvaluate & Fragment)*	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
PointLightShader *(Vertex & Fragment)*	Point Light-Beleuchtung
NullShader *(Vertex & Fragment)*	Für den Stencil Test
SkyShader *(Vertex & Fragment)* (Aufg 1)	Zeichnet die Skymap im Hintergrund
BlurShader *(Vertex & Fragment)*	Shader, für den Blureffekt (mit advanced 2-Pass Filter; es reicht ein Shader mit einer entsprechenden Abzweigung)
PostProcessShader *(Vertex & Fragment)*	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) lightVisShader *(Vertex & Fragment) [optional]*	Kann Text auf dem Display ausgeben Zeichnet die Punkt-Lichtquelle als Primitive Geometrie (Cube oder Sphere)

* Hier findet das Normalmapping und Parallaxmapping statt.





Tipps:

1) Zum einlesen von JSON Dateien, kann der in dem Material bereitgestellte JSON Parser verwendet werden. Es können aber bei Bedarf auch andere 3rd Party APIs oder eigene genutzt werden.
2) Die Funktion `texelFetch` arbeitet nicht mit Texturkoordinaten im Interval [0,1] auf beiden Achsen, sondern mit Texelkoordinaten in den Intervallen [0, Texturbreite] und [0, Texturhöhe]. Die Variable `gl_FragCoord` enthält die Position des Fragments im Screen Space. Diese kann direkt für den Zugriff verwendet werden.
3) *glBlitFramebuffer(...)* kopiert Pixel von einem (Read) Framebuffer in einen anderen (Draw) Framebuffer. Es können die Source- und Destination Rectangles mit den Parametern angegeben werden, eine Filtermethode sowie ein Filter was passieren soll, sollten die Framebuffer nicht dieselbe Größe besitzt. Ebenfalls muss mit der Maske angegeben werden, was wir eigentlich kopieren wollen. (Hat ein FB mehr Targets (Color-Attachments), muss hier auch der Richtige aktiviert sein!). Es können mit dieser Operation auch Depth- oder Stencil-Buffer kopiert werden.
4) Wenn mittels glBlitFramebuffer Teile eines Buffers in einen anderen kopiert werden sollen, muss neben dem Binden des FBO das Attachment als Buffer gewählt werden.
   glReadBuffer(GL_COLOR_ATTACHMENTX)
   Wenn das Attachment eine Textur ist, kann diese als mittels der ID (generiert bei glGenTexture(1, &id)) als Textur gesetzt werden.
   glActiveTexture(GL_TEXTURE0);
   glBindTexture(GL_TEXTURE_2D, id);
5) Die Unit light.h enthält bereits Structs für verschiedene Lichtquellen, um den Code gut modularisiert & kompakt zu halten, können hier Methoden wie das Setzen von Uniformvariablen etc. ergänzt werden. Dieser befindet sich ebenfalls im Abschnitt Material.
   In diesem befinden sich auch die in der Musterlösung verwendeten Attentuation Werte.
6) Die weiteren Parameter aus der Unit material.h wie shininess oder der Diffus/Albedo vec3 sowie die Shininess müssen nicht verwendet werden und dürfen in der Beleuchtungsberechnung im entsprechenden Shader Konstant gesetzt werden.
7) Die Unit gBuffer enthält ein Aufzählungstyp für die G-Buffer Texturen, sowie ein Struct für den G-Buffer und Methoden Deklarationen sowie Beschreibungen zu diesen. Um die Aufgabe incl. aller weiteren Anforderungen umzusetzen, muss ggf. das Strukt angepasst werden und oder die Methoden verändert/ergänzt werden. Die Methoden und Beschreibungen dienen zu Hife um die Kernanforderungen hinzubekommen. (Etwas Denkarbeit ist gerade beim Bloom-Effekt gefordert)
8) Tessellation muss für die Szene aufgrund der fehlenden Heightmap und aus Effizienzgründen nicht durchgeführt werden. Für das Quad jedoch weiterhin.
9) Zur Einfachheit muss der **Wireframe**-Modus nicht umgesetzt werden. (Kann jedoch als Herausforderung gerne umgesetzt werden)! Ebenfalls muss der mit dem Geometrieshader visualisierte Tangentspace nicht mehr angezeigt werden.
10) Die Textur Transformation in den linear Space kann beim einlesen der Textur als sRGB durchgeführt werden oder im GBuffer Shader manuell.

Weitere (freiwillige) Funktionalitäten:

1) Es können weitere **Post-Processing Effekte** (z.B. Kernelfilter-Effekt/Negativ) auf die finale Ausgabe im Fenster angewendet werden. Hierbei kann mittels Filterung das Bild manipuliert werden (Blur) oder Farbkorrekturen (Schwarz-Weiß, Farbbereich Anpassung/LUT) durchgeführt werden (Welche Effekte Ihr implementiert, ist euch überlassen. Es sollte jedoch die Theorie dazu verstanden sein, sowie die Sinnhaftigkeit in Spielen/Simulationen erklärt werden können). Diese sollen jeweils per Tastendruck umschaltbar sein.
2) Es kann per Tastendruck ein Look-Up Table als Post Processing Effekt verwendet werden.
3) Erstellen einer/mehrerer Spotlichtquelle(n) mit entsprechenden Bounding-Volumen. (Cylinder [oder Hemisphere mit Orientierung])
4) Wireframe, hier muss überlegt werden, welche Teile als Wireframe gerendert werden sollen und welche nicht (da man bei kompletter Aktivierung ggf. nur ein Wireframe-Rechteck am Ende erhält).
5) Der Szene kann etwas mehr "Leben" eingehaucht werden, indem pro Frame die Positionen der Punktlichtquellen um einen zufälligen normierten Offset-Vektor versetzt werden.
6) Es können weitere **Modelle in die Szene geladen** werden.
7) Es können per Tastendruck zufällige in der Szene verteilte Lichtquellen erzeugt werden. Die Anzahl ist per Tastendruck variabel.