Assignment 3B, Deadline 22.12.2023 um 23:59

Wenn Sie das Projekt starten, sollten Sie eine Menge bunter Spheres sehen. Diese wurden mit derselben Codebasis erzeugt, die wir auch in Assignment 2 benutzt haben – einziger Unterschied: man kann jetzt auch die Projektionsmatrix mit **`OglHelper::setProjectionMatrix`** setzen, da wir uns mit dieser jetzt auch schon befasst haben, und weil Sie diese Matrix fürs Assignment auch benutzen müssen.

Was Sie jetzt implementieren sollen, ist ***Object Picking***: wenn mit der linken Maustaste auf eine Sphere geklickt wird, soll diese selektiert werden – Sie zeigen das an, indem Sie die Sphere rot einfärben. Solange die linke Maustaste gedrückt bleibt, soll die selektierte Kugel dann der Maus folgen. (Siehe Video) Wenn die linke Maustaste losgelassen wird, wird auch die Sphere de-selektiert.

Für die Maus-Interaktion gibt es zwei relevante Funktionen, **`App::mousePosition`** (wird einmal pro Frame aufgerufen, sofern sich die Maus überm Fenster bewegt & sagt Ihnen die derzeitigen Pixel-Koordinaten des Mauszeigers) sowie **`App::mouseButton.`** (wird aufgerufen, wenn sich die Maus überm Fenster befindet & eine Maustaste gedrückt wird)

Das Problem ist sehr ähnlich wie Raytracing zu lösen: Wenn die linke Maustaste gedrückt wird, müssen Sie einen Strahl definieren, der an der Kameraposition startet und durch die aktuelle Mausposition verläuft. Sie testen diesen Strahl auf Intersection mit den Spheres in der Szene – falls hier mehrere Spheres intersected werden, selektieren Sie diejenige, die der Kamera am nächsten liegt. Wenn Sie nun via **`App::mousePosition`** benachrichtigt werden, dass sich die Maus bewegt und gleichzeitig eine der Spheres selektiert ist, dann updaten Sie die Position der Sphere entsprechend.

@Intersection: Sie könnten die Ray-Sphere Intersection natürlich lösen, indem Sie den Strahl vs. alle Triangles einer Sphere testen, den Code dafür hätten Sie vom Raytracing Assignment schon. Das wäre aber nicht sehr performant; Für einfache Geometrien wie Spheres existieren eigene, optimierte Intersection Tests. Sie können z.B. dieses Tutorial nutzen: https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-sphere-intersection.html, oder sich selbst ein Tutorial suchen, es gibt dazu so viele. (Geben Sie bitte im Code die Quelle an.)

@Koordinatensysteme: Die Challenge in dieser Aufgabe liegt in den Koordinatensystemen. Die Kameraposition und die Position der Spheres sind in Weltkoordinaten gegeben; Die Mausposition in Pixelkoordinaten. Sie müssen also aus der 2D Pixelkoordinate eine 4D Screen-Space Koordinate machen (gehen Sie davon aus, dass der Mausklick entweder an der Near Plane oder der Far Plane passiert ist, und wählen Sie den Z-Wert entsprechen) und diese dann mit Hilfe der Viewport, Projection und View Matrizen in den World Space umrechnen. Die Formel für die Viewport Matrix haben Sie auf den LVA-Slides, bzw. habe ich diese Matrix auch für den C++ Rasterizer selbst bauen müssen. (Nicht vergessen: das Speicherlayout von GLM-Matrizen ist column-major und nicht row-major)

Ich weiß, wir haben für genau dasselbe Problem beim Raytracing eine geometrische Lösung hergeleitet; Es geht bei der Aufgabe aber ganz konkret um die Verwendung von Matrizen und Koordinatenräumen.

Beim Verschieben der Sphere müssen folgendes beachten: Sie können die Sphere nicht einfach so auf die Position der Maus setzen: Sie möchten zwar die Pixel-Position der Maus übernehmen, aber die Distanz der Sphere zur Kamera – also den Z-Wert - möchten Sie nicht ändern. Sie müssen also

den Z-Wert der Sphere berechnen; Den bekommen Sie, indem Sie die Sphere-Position von Welt in Screen-Koordinaten umrechnen. Wenn Sie den Z-Wert haben, kombinieren Sie ihn mit den Pixel-Koordinaten des Mauszeigers, um die aktualisierte Position in Screen Koordinaten zu bekommen, und rechnen diese dann wieder auf Weltkoordinaten um.

Schreiben Sie am besten 2 Hilfsfunktionen:

- Eine, die Weltkoordinaten unter Verwendung von View/Projection/Viewport Matrizen in Screen Koordinaten umrechnet.
- Und eine, die Screen Koordinaten unter Verwendung von View/Projection/Viewport Matrizen in Weltkoordinaten umrechnet.

Hier nochmal die wichtigsten Regeln fürs Anwenden von Transformationsmatrizen auf Vertices:

- Sie hängen 1.f als 4te Komponente an einen Vertex, um mit 4x4 Matrizen multiplizieren zu können.
- Wenn die 4x4 Matrix affin ist, können Sie die 4te Komponente nach der Multiplikation einfach wieder wegstreichen. (weil immer noch 1)
- Wenn die 4x4 Matrix nicht affin ist, müssen Sie erst den Vertex durch die 4te Komponente dividieren, bevor Sie die 4te Komponente wieder wegstreichen können.
- Wenn sie mehrere 4x4 Matrizen miteinander multiplizieren und mindesten eine davon ist nicht affin, dann ist auch die kombinierte Matrix nicht affin.
- Wenn Sie aber nur affine Matrizen miteinander multiplizieren, dann ist die kombinierte Matrix immer noch affin.
- Die Inverse einer 4x4 Transformationsmatrix stellt immer die inverse Transformation dar – das gilt auch für die Projection Matrix. Sie können also z.B. mit der inversen View Matrix vom View Space in den World Space wechseln.
- Die Inverse einer affinen Matrix ist affin.
- Die Inverse einer nicht affinen Matrix ist nicht affin. (→ d.h. nach der Anwendung brauchts auch wieder eine Division durch w)