entwickler.press
shortcuts

Erfolgreiche Spieleentwicklung

Water-, Terrain- und GUI-Rendering

Alexander Rudolph

Alexander Rudolph

Erfolgreiche Spieleentwicklung

Water-, Terrain- und GUI-Rendering

ISBN: 978-3-86802-539-2

Ein Imprint der Software & Support Media GmbH

Quellcodebeispiele

Alle hier nicht gezeigten Listingbeispiele finden Sie aus Platzgründen unter https://entwickler.de/press/erfolgreiche-spieleentwicklung-7-128048.html unter „Listings zum shortcut“.

1 Wasserspiegelungen, 3-D-Wellen, Schaumkronen und Kaustiken als Post-Processing-Effekt

Realistische Echtzeitwassereffekte sorgen seit jeher für den berüchtigten Wow-Effekt in der Spielercommunity. In diesem Kapitel tauchen wir gemeinsam in die Tiefen des Water-Renderings hinab und untersuchen, wie sich 3-D-Wellen, Spiegelungen auf der Wasseroberfläche, Schaumkronen sowie Unterwasserkaustiken im Verlauf der Post-Processing-Phase simulieren lassen.

Tropische Inseln, traumhafte Sandstrände und glasklares Wasser – nie zuvor wurde das „Paradies auf Erden“ so stimmungsvoll in Szene gesetzt wie in Far Cry, einem der besten Spiele des Jahres 2004. Gemessen am heutigen Stand der Technik wirkt besagtes Spiel mittlerweile zwar ein wenig angestaubt, damals jedoch kam man als Spieler aus dem Staunen nicht mehr heraus. Die grafische Qualität und der Detailreichtum der Spielewelt waren bis dato unerreicht, wobei insbesondere die Wasserdarstellung hoch gelobt und als richtungsweisend angesehen wurde. Dreidimensionale Wellenbewegungen gab es zwar noch nicht zu bestaunen, dafür jedoch in Echtzeit berechnete Spiegelungen auf der

Wasserfläche, Refraktionseffekte (Lichtbrechung), Unterwasserkaustiken sowie Brandungswellen.

Herausforderungen und Schwierigkeiten bei klassischer Wasserdarstellung

Bevor wir uns gleich ausführlich damit befassen, inwieweit die Wasserdarstellung (**Abb. 1.1**, **1.2** und **1.3**) von dieser neuartigen Technik (die Durchführung der Beleuchtungs- und Post-Processing-Berechnungen erfolgt zeitlich verzögert nach Abschluss der Geometrieverarbeitung) profitieren kann, sollten wir zunächst mit einigen Worten auf die Schwierigkeiten eingehen, die im Rahmen des klassischen Forward Renderings (Geometrieverarbeitung und Beleuchtungsberechnungen erfolgen hierbei zeitgleich) zu bewältigen sind.

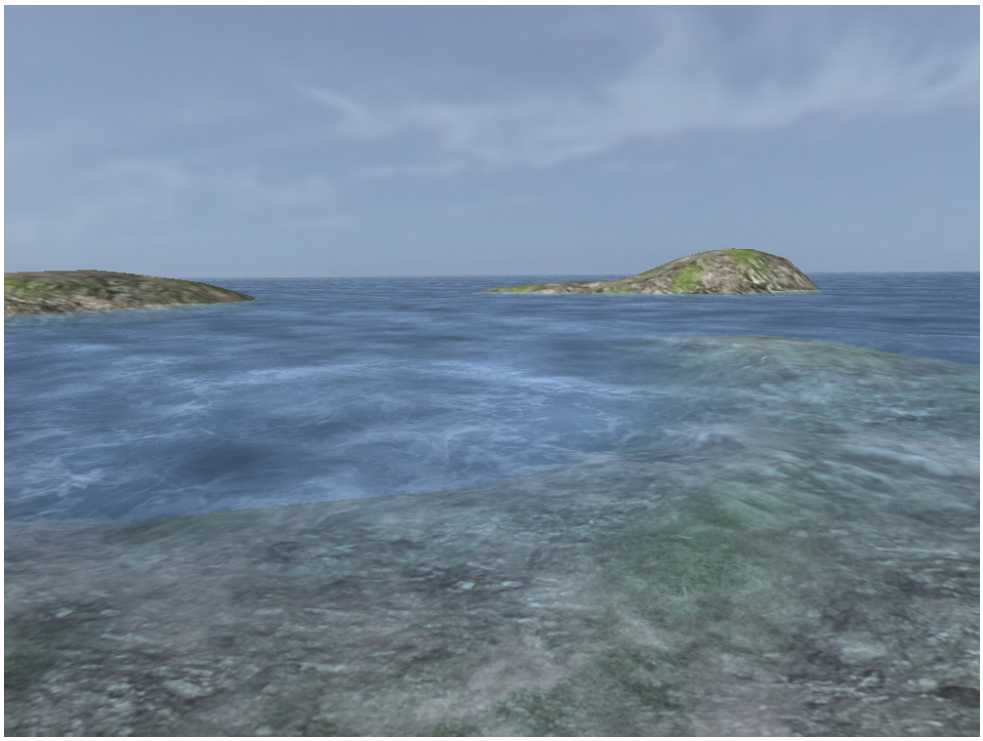

Abbildung 1.1: Wasserdarstellung (OpenGL-Framework-Demoprogramm 35)

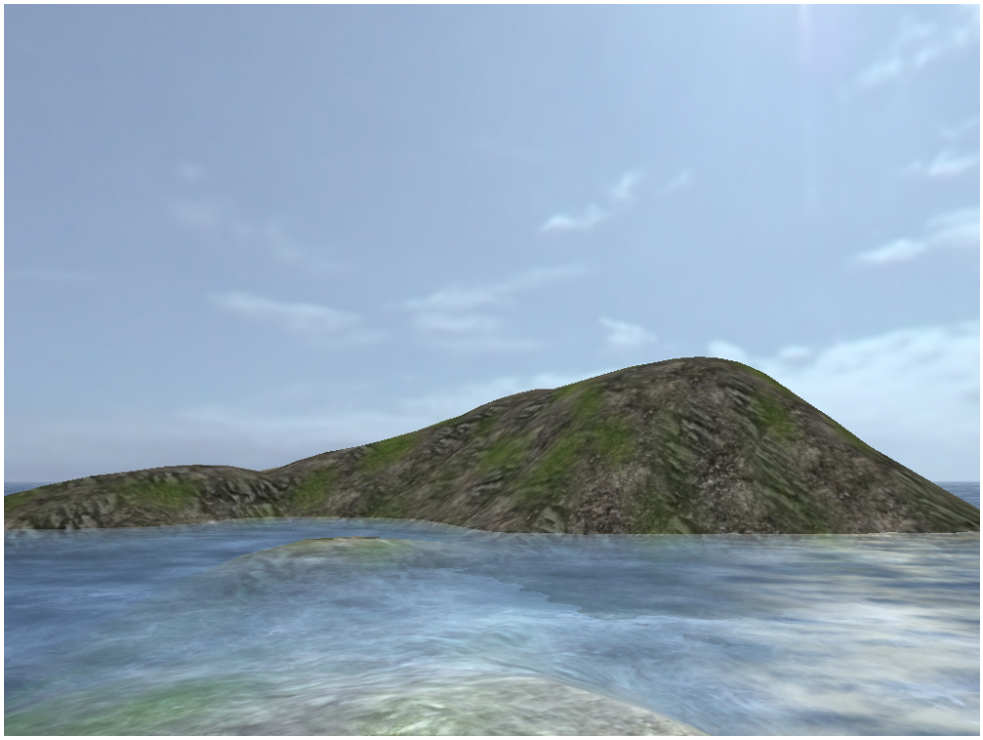

Abbildung 1.2: Wasserdarstellung

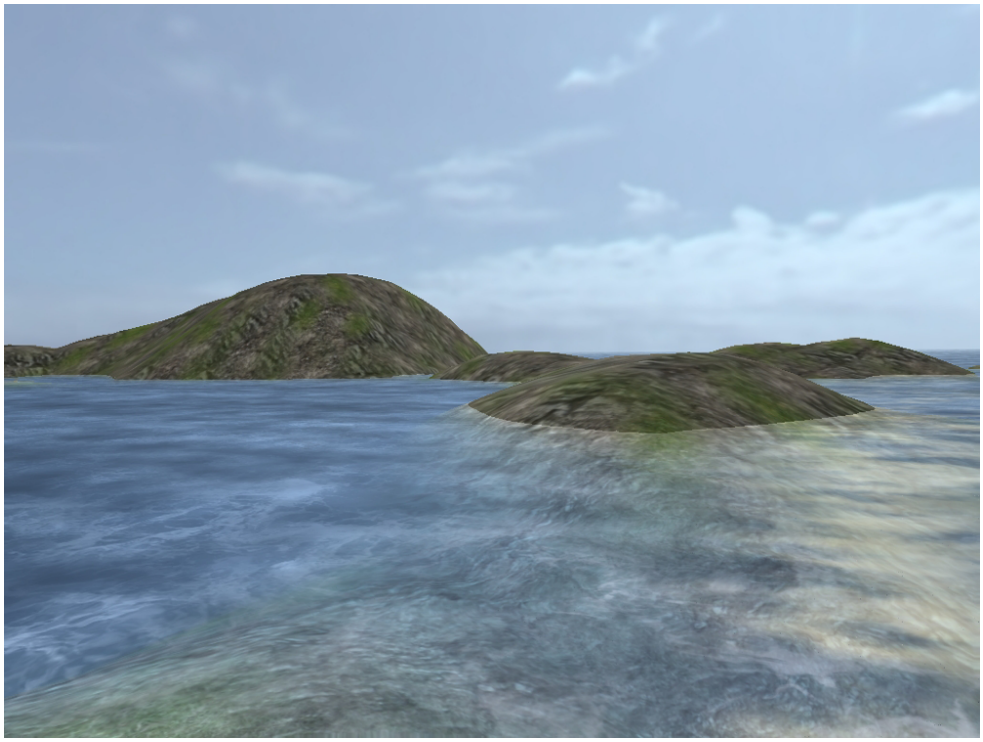

Abbildung 1.3: Wasserdarstellung

Zu Beginn des neuen Millenniums kamen bei der Wasserdarstellung lediglich blaue, halbtransparente Polygonflächen zum Einsatz, die dann einige Zeit später mithilfe von Texturanimationen optisch aufgewertet wurden. Für die Simulation der physikalischen Eigenschaften des Wassers war die Leistung der damaligen Prozessoren und Grafikkarten jedoch bei Weitem nicht ausreichend. Die besagten Wassereffekte wirkten primitiv und ließen sich ohne viel Aufwand implementieren – es war lediglich erforderlich, im letzten Schritt der Szenendarstellung alle sichtbaren halbtransparenten Wasserflächen bei aktiviertem Color Blending zu rendern. Wollte man jedoch darüber hinaus die grundverschiedenen Lichtverhältnisse an Land bzw. im Wasser berücksichtigen, so wurden die Dinge schlagartig komplizierter. Die Tatsache, dass man in diesem Zusammenhang für die Szenengeometrie unterhalb und oberhalb der Wasseroberfläche unterschiedliche Beleuchtungsberechnungen durchführen musste, machte es erforderlich, die Spielewelt in zwei Schritten zu rendern und die Wasserfläche in diesem Zusammenhang als Clipping Plane zu verwenden. Wie dem auch sei, die verbesserten Beleuchtungsberechnungen führten jedoch nicht dazu, dass die Wassereffekte in den damaligen Spielen auch nur ansatzweise wie echtes Wasser wirkten. Dies sollte sich erstmals mit der Darstellung von in Echtzeit berechneten Wasserspiegelungen ändern. Abermals verkomplizierte sich die Situation, denn hierfür war es nicht nur erforderlich, die gespiegelte Szenengeometrie von einer anderen Blickrichtung und einer anderen Position aus in einem separaten Render Target (Render to Texture) zwischenzuspeichern; fehlerhafte Wasserspiegelungen ließen sich darüber hinaus nur mithilfe zusätzlicher Sichtbarkeitstests vermeiden. Wie genau sich besagte Positionen und Blickrichtungen für diesen zusätzlichen Render-Durchgang ermitteln lassen, werden wir an dieser Stelle nicht im Detail erörtern. Stattdessen belassen wir es bei einem einfachen Beispiel: Blickt man als Spieler senkrecht auf die

Wasseroberfläche, dann weist die gespiegelte Blickrichtung senkrecht zum Himmel. Befindet man sich als Spieler einen Meter über dem Wasser, dann befindet sich die gespiegelte Position einen Meter unterhalb der Wasseroberfläche.

Den Wellengang simulierte man zunächst ausschließlich mittels einer Echtzeitmodifikation der Normalenvektoren der Wasseroberfläche (keine echten 3-D-Wellen), was zu einer verzerrten Darstellung der Wasserspiegelungen führt. Die tägliche Erfahrung lehrt uns jedoch, dass sich die Wellenbewegungen nicht nur auf das Erscheinungsbild der Reflexionen auswirken; Gleiches gilt für die Wahrnehmung aller Objekte, die sich auf der anderen Seite der Wasserfläche befinden. So hat ein Beobachter außerhalb des Wassers einen ähnlich verzerrten Blick auf die Unterwasserwelt wie ein Taucher auf die Welt über dem Wasser. Verantwortlich hierfür ist das gleiche optische Phänomen, dem wir auch die Regenbogen am Himmel zu verdanken haben – die Lichtbrechung bzw. Refraktion, welche immer am Übergang zweier optisch unterschiedlich dichter Medien (z. B. Luft und Wasser) auftritt. Wie in **Abbildung 1.4** gezeigt, werden die Lichtstrahlen beim Übergang von der Luft (optische Dichte = 1) ins Wasser (optische Dichte = 1,333) zum Normalenvektor der Wasserfläche hin gebrochen, wobei die Änderung der Ausbreitungsrichtung bei kurzwelligem blauen Licht deutlich stärker ausgeprägt ist als bei langwelligem roten Licht (Dispersion, „Regenbogeneffekt“). Tritt ein Lichtstrahl hingegen aus einem optisch dichteren Medium aus, so wird dieser vom Normalenvektor der Grenzfläche weggebrochen. Quantitativ beschreiben lässt sich dieses Verhalten mithilfe des Brechungsgesetzes nach Snellius (**Abb. 1.4**).

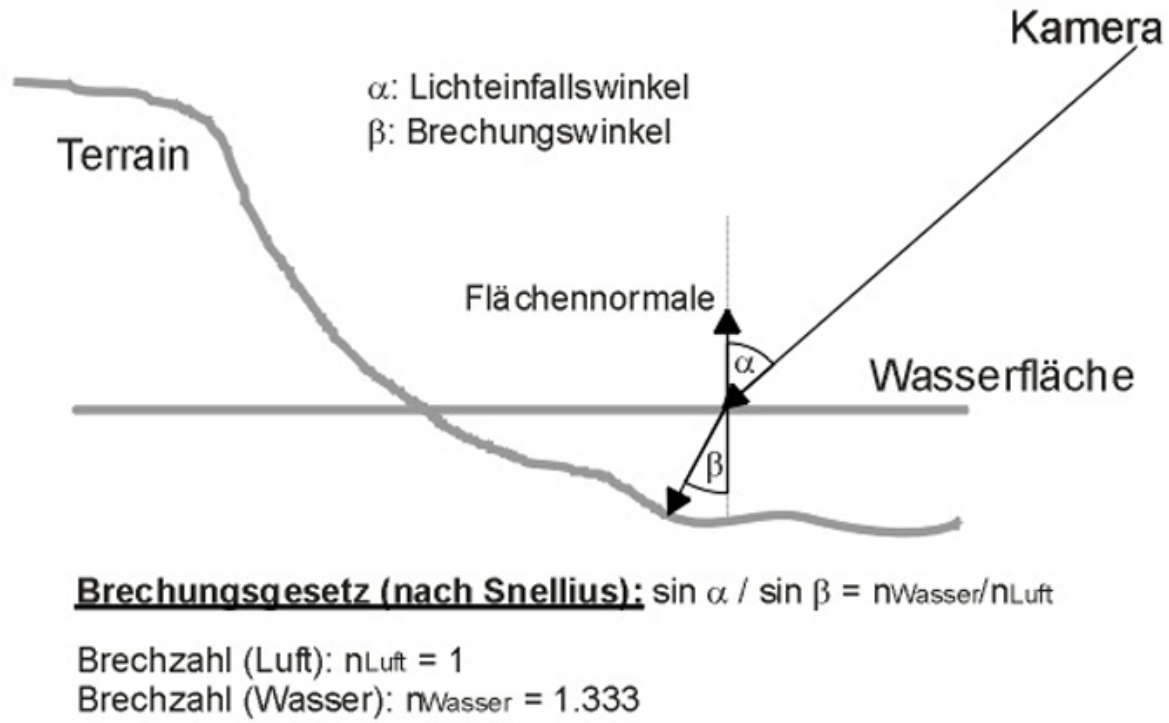

Abbildung 1.4: Simulation der Lichtbrechung beim Luft-Wasser-Übergang

Da die Darstellung dieser Refraktionseffekte mithilfe des bereits zur Sprache gekommenen Color Blendings nicht möglich ist, müssen wir stattdessen ähnlich wie bei der Berechnung der Wasserspiegelungen entweder die Unterwasserszenerie (sofern sich der Beobachter bzw. die Kamera über dem Wasser befindet) oder aber die 3-D-Szene außerhalb des Wassers (sofern sich die Kamera unter Wasser befindet) in einem weiteren Render Target zwischenspeichern. Hier nun eine Übersicht über alle Schritte, welche zur Darstellung einer Wasserfläche samt Spiegelungen und Refraktionseffekten erforderlich sind.

Fall 1 – Kamera über dem Wasser

- Im ersten Schritt wird die an der Wasseroberfläche gespiegelte 3-D-Szene in ein zusätzliches Render Target (Reflection Map) gezeichnet (Render to Texture).

- Im zweiten Schritt wird die Unterwasserwelt in ein zweites Render Target (Refraction Map) gezeichnet.
- Im dritten Schritt erfolgt die Wasserdarstellung, in deren Verlauf die zuvor generierten Texturen auf die Wasserfläche gemappt werden (Projective Texturing/Reflection bzw. Refraction Mapping).
- Im letzten Schritt wird schließlich die 3-D-Szene über dem Wasserspiegel gerendert.

Fall 2 – Kamera im Wasser

- Im ersten Schritt wird die 3-D-Szene oberhalb des Wasserspiegels in ein zusätzliches Render Target (Refraction Map) gezeichnet.
- Im zweiten Schritt erfolgt die Wasserdarstellung, in deren Verlauf die zuvor erstellte Refraction Map auf die Wasserfläche gemappt wird.
- Im letzten Schritt wird schließlich die Unterwasserwelt gerendert.

Als wäre das alles nicht schon kompliziert genug, wartet auf uns bereits die nächste Herausforderung – die Simulation von dreidimensionalen Wellenbewegungen. Während sich ebene Wasserflächen mithilfe einfacher Vertex-Quads darstellen lassen, erfordert die Visualisierung von realistisch wirkenden Wasserwellen ein engmaschiges Vertexgitter, welches noch dazu in Echtzeit animiert werden muss. Neben der offenkundigen Frage, wie sich eine solch große Anzahl von Wasser-Vertices ohne einen signifikanten Einbruch der Frame Rate animieren und rendern lässt, steht man als Programmierer noch vor einem ganz anderen Problem, für welches es im Rahmen der klassischen Wasserdarstellung leider keine saubere Lösung gibt: Sämtliche unserer Reflexions- und Refraktionsberechnungen liefern streng genommen nur bei ebenen Wasserflächen ein korrektes Ergebnis, da sich bei einer welligen Oberfläche die Lage der Reflexionsebene bzw.