Proceduralne generowanie planet w czasie rzeczywistym
w dynamicznie zmieniającej się skali

Dokumentacja techniczna

MATEUSZ CHECHLIŃSKI

Wersja 4.0

31 stycznia 2015

Modyfikacje dokumentu			
Wersja	Data	Autor	Opis
1.0	27 listopada 2014	Mateusz Chechliński	Iteracja nr 1.
2.0	12 grudnia 2015	Mateusz Chechliński	Iteracja nr 2.
3.0	10 stycznia 2015	Mateusz Chechliński	Iteracja nr 3.
4.0	31 stycznia 2015	Mateusz Chechliński	Iteracja nr 4.

Spis treści

1. Słownik pojęć

Shader - krótki program komputerowy, który oblicza właściwości pikseli oraz wierzchołków.

Uniform - globalna zmienna GLSL zadeklarowana ze ze słowem kluczowym „uniform". Wykorzystywana w roli atrybutu przekazywanego do shadera.

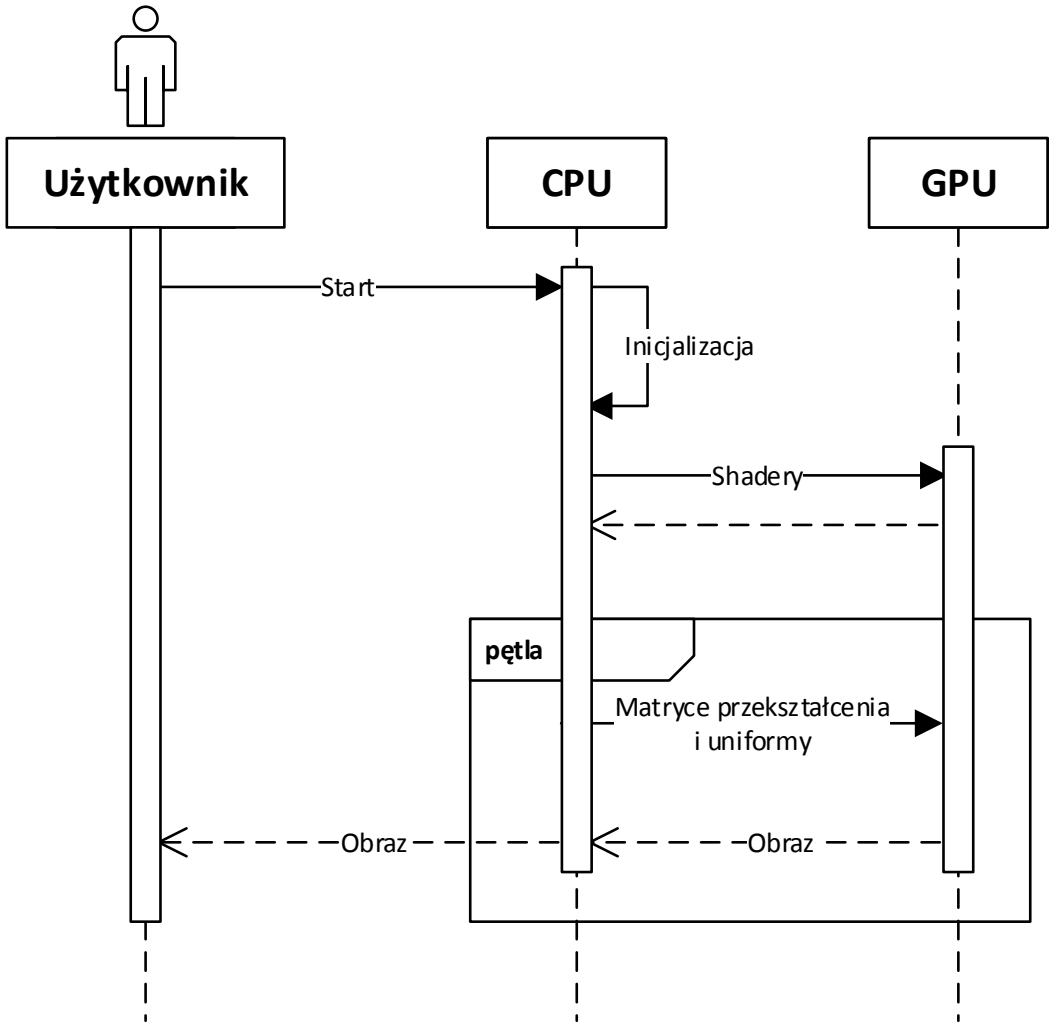
Renderowanie - (od ang. *rendering*), analiza modelu danej sceny oraz utworzenie na jej podstawie dwuwymiarowego obrazu wyjściowego w formie statycznej lub animacji.

2. Wstęp

Niniejszy dokument stanowi dokumentację techniczną aplikacji do proceduralnego generowania planet w czasie rzeczywistym w dynamicznie zmieniającej się skali.

3. Iteracja nr 1

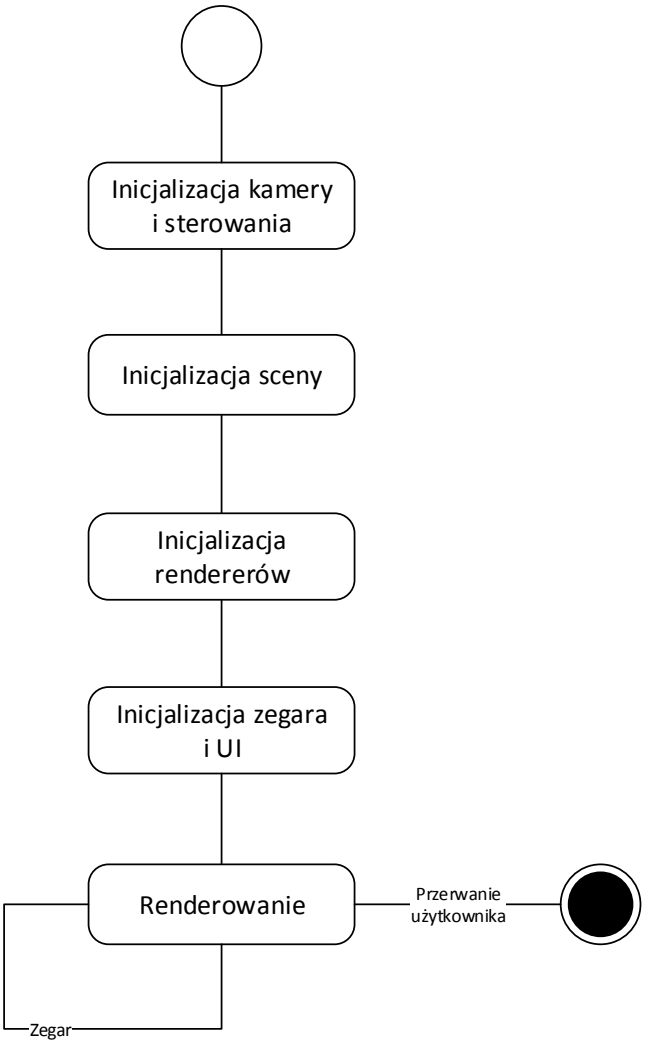
3.1 CPU - GPU



Rys. 1 - Schemat komunikacji CPU - GPU

Po uruchomieniu aplikacji (otwarciu strony w przeglądarce) wykonana zostanie inicjalizacja po stronie CPU. Potem załadowane do GPU zostaną shadery. Wówczas rozpoczyna się pętla, w której CPU przekazuje do GPU matryce przekształceń oraz uniformy, a GPU renderuje na tej podstawie obraz, który następnie jest prezentowany użytkownikowi.

3.2 Inicjalizacja



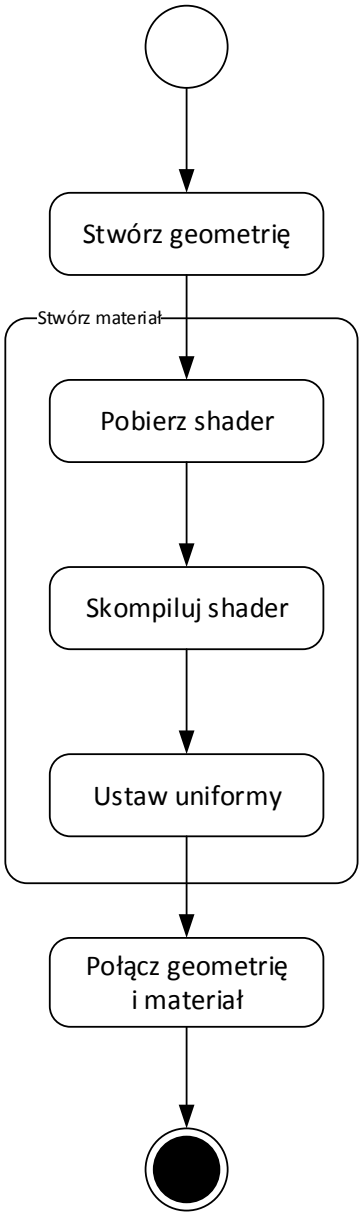
Rys. 2 - Inicjalizacja aplikacji

Powyższy schemat przedstawia kolejność, w jakiej inicjalizowane są poszczególne elementy systemu.

W tej części, spore zastosowanie ma biblioteka Three.js, która pozwala całą inicjalizację zawrzeć w zaledwie kilku liniach kodu.

Po zakończeniu inicjalizacja, aplikacja przechodzi do pętli renderującej, która przerwana może zostać tylko poprzez zakończenie pracy aplikacji.

3.3 Generowanie planet i gwiazd



Rys. 3 - Schemat generowania planet i gwiazd

Powyższy schemat przedstawia kolejne kroki związane z wygenerowaniem planety lub gwiazdy. W pierwszej kolejności tworzona jest geometria (sfera). Następnie pobierane i ładowane do GPU są shadery (Vertex i Fragment Shader), po czym ustalane są wartości uniformów (co do reguły, ich wartość nie będzie się zmieniać - poza uniformem czasu).

Następnie tak utworzone dane są łączone i tworzą planetę lub gwiazdę.

3.4 Metody

3.4.1 PlanetManager

Metoda	Opis
startTime()	Oblicza Time Uniform
planetTypes	Słownik trzymający dane o dostępnych typach planet
updateAnimatedUniforms(material, animatedUniforms)	Metoda odświeżająca uniformy oparte na funkcjach
createMaterial(data)	Tworzy materiał na podstawie danych. Kopiuje materiał zawierający shadery oraz zapełnia ich uniformy.
createPlanet(name)	Tworzy planetę wybranego typu (lub gwiazdę)
update()	Odświeża planety

3.4.2 Resouces

Metoda	Opis
getShaderMaterial(name)	Pobiera oba shadery razem
getTexture(name)	Pobiera plik tekstury

3.4.3 StaticLoader

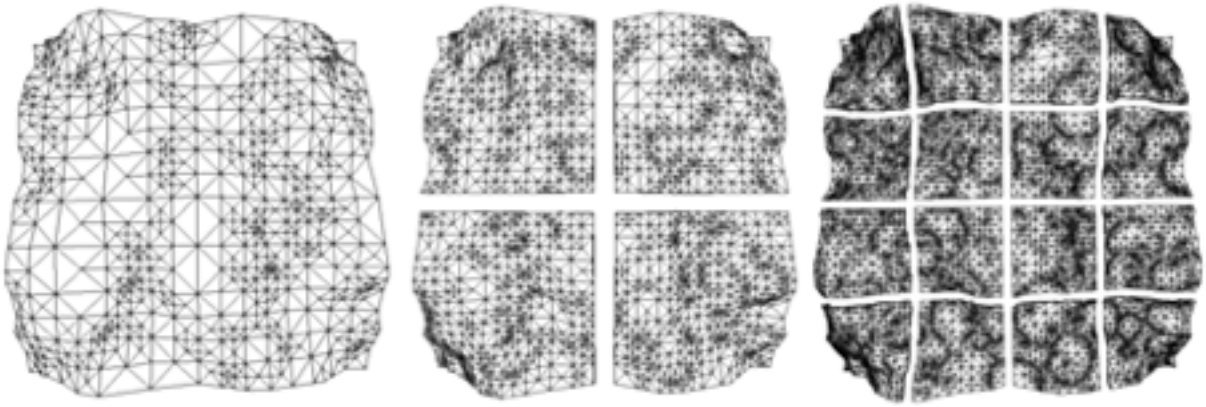
Metoda	Opis
ajaxGet(path)	Wysyła zapytania AJAX
loadShader(name)	Pobiera wskazany shader

4. Iteracja nr 2

Celem drugiej iteracji było zaimplementowanie algorytmu *Level of Details* (poziom szczegółów). Po analizie porównawczej popularnych algorytmów LOD (m.in. SOAR, ROAM, Progressive Meshes) zdecydowaliśmy się na implementację algorytmu Chunked LOD.

4.1 Chuncked LOD

Ten opracowany przez Thatchera Urlicha algorytm, podobnie jak SOAR, wykorzystuje strukturę drzewa czwórkowego. Początkowa kwadratowa siatka, z każdym zejściem o jeden poziom niżej, dzielona jest na cztery równe części, przy czym każda z tych części ma taką samą ilość wierzchołków co rodzic.



Trzy kolejne poziomy szczegółów w algorytmie Chunked LOD

W odróżnieniu od pozostałych, algorytm Chunked LOD nie stara się uzyskać optymalnego podziału siatki, lecz skupia się na uzyskaniu jak największej „przepustowości" przy minimalnym obciążeniu CPU. Podstawą jego działania jest fakt, że współczesne GPU są na tyle wydajne, że szybciej jest wyrenderować nieco więcej trójkątów niż wykonać pewną ilość dodatkowych obliczeń na CPU. Pomysł polega na zastosowaniu poziomu szczegółów zależnego od obserwatora nie do pojedynczych wierzchołków, a do większych segmentów (ang.: chunks).

Drzewo czwórkowe jest budowane w fazie preprocessingu, a następnie ładowane do pamięci. Za każdym razem, gdy teren ma być wyrenderowany, algorytm schodzi rekurencyjnie po drzewie czwórkowym wyznaczając odpowiedni poziom szczegółów dla danego kwadrata. Poziom szczegółów sąsiadujących części może się różnić.

W naszej aplikacji faza preprocessingu następuje nie przed uruchomieniem aplikacji, lecz za każdym razem, gdy funkcja określająca poziom szczegółów zwróci wynik informujący o konieczności zmiany liczby detali.

4.2 Reprezentacja sfery

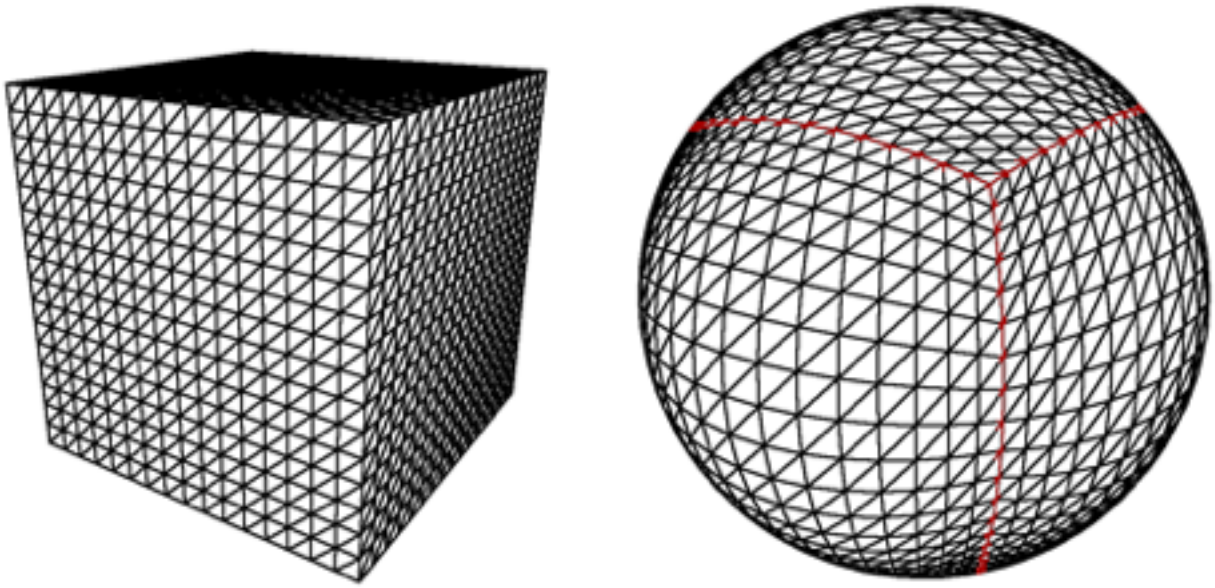
W pierwszej wersji projektu planeta była reprezentowana za pomocą klasycznej siatki geograficznej, lecz szybko okazało się, iż rozwiązanie to nie nadaje się z kilku powodów.

Po pierwsze, o ile implementacja tego rozwiązania była prosta, gdy wysokość każdego wierzchołka była obliczana w każdym przebiegu pętli czasu, o tyle nie dawało się w prosty sposób (bez przekłamań) nanieść na tak reprezentowaną sferę mapy wysokości wyliczanej raz i zapisywanej w pamięci.

Po drugie, zastosowanie algorytmu LOD wymaga by siatkę bryły dzielić na mniejsze części w zależności od odległości danego fragmentu od obserwatora. Dzielenie tak reprezentowanej sfery

na równe części nie jest łatwe ani wydajne, więc konieczna stała się zmiana stosowanego podejścia.

Wykorzystaliśmy technologię QuadCude, w której sfera jest rzutowana na sześcian, na którym jest opisana, za pomocą perspektywy krzywoliniowej. Sfera podzielona jest na sześć równych części i każda z nich odpowiada jednej ścianie sześcianu.



Sześcian i jego sferyczna projekcja

Zaletą tej projekcji jest to, iż nie występują na niej osobliwości (punkty w których wartość jest nieokreślona) na biegunach czy gdziekolwiek indziej. Sześcian można w łatwy i szybki sposób rzutować na sferę. Równie łatwe jest nakładanie mapy wysokości na kwadratową ścianę, a także dzielenie każdego kwadratu na cztery mniejsze, co ma ogromne znaczenie dla poziomu szczegółów.

4.3 Mapa wysokości

Wysokość każdego punktu nad „poziomem 0" powierzchni planety jest określana za pomocą mapy wysokości (planeta na „poziomie 0" jest idealną, z dokładnością do rozdzielczości siatki, sferą, natomiast jej powierzchnia jest pofałdowana).

W poprzedniej iteracji wysokość każdego wierzchołka była określana w czasie rzeczywistym, za każdym przejściem pętli czasu. Ze względu na wydajność podjęliśmy decyzję by zrezygnować z tego na rzecz generowania map wysokości tylko wtedy, gdy będzie to konieczne, tj. gdy funkcja oceny poziomu szczegółów określi, że należy poziom szczegółów zmienić.

Mapa wysokości to tekstura w odcieniach szarości, w której kolor każdego piksela określa wysokość korespondującego wierzchołka (ang.: vertex) - kolor biały to 1.0 czyli maksymalna wysokość, zaś kolor czarny to 0.0 czyli wysokość minimalna („poziom 0"). Kolor piksela wyznaczany jest jako suma ważona pewnej ilości oktaw szumu Simplex.

Do generowania map wysokości wykorzystujemy technologię zwaną z ang. „**Offscreen rendering**", która polega na renderowaniu do tekstury itp. zamiast na ekran. Jej implementacja w niniejszej pracy dyplomowej wygląda następująco:

- Tworzona jest nowa scena, która zawiera ortogonalną kamerę i siatka (ang.: mesh);

- Siatka jest podzielona na kwadraty, w rogach których ustawiane są odpowiednie atrybuty, kamera zaś jest ustawiona naprzeciwko siatki.

- Shader w pierwszym przebiegu renderuje na podstawie uprzednio ustawionych atrybutów obraz, który zapisywany jest w teksturze;

- Początkowo siatka zawierała tylko jeden kwadrat naraz i renderowano tylko jedną oktawę, lecz okazało się, iż rozwiązanie to jest mało wydajne. Wobec tego siatka podzielona jest na wiele kwadratów, w następujący sposób:

- Pojedynczy kwadrat to jedna oktawa szumu;

- Pojedynczy wiersz kwadratów to szereg kolejnych oktaw, które następnie zostaną do siebie dodane (suma ważona);

- Kolejne wiersze odpowiadają kolejnym mapom wysokości;

W drugim przebiegu, shader „tnie" siatkę na poszczególne kwadraty, a następnie w każdym z wierszy sumuje je z odpowiednimi wagami. Tak otrzymane mapy wysokości nanoszone są na bryłę planety, dając wrażenie naturalnie pofałdowanej powierzchni.

5. Iteracja nr 3

W tej iteracji celem było wykorzystanie dotychczasowych efektów pracy w celu generowania całego kosmosu, czyli wielu gwiazd i planet krążących wokół nich.

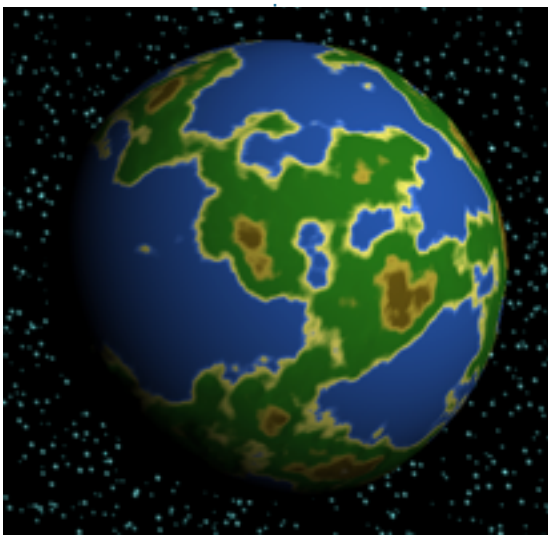
5.1 Rodzaje planet

Aby kosmos był ciekawy, opracowaliśmy kilka rodzajów planet, przy każdym z nich starając się, by posiadał pewną charakterystyczną cechę. Listę rodzajów zebrano poniżej:

5.1.1. Ziemio-podobna

Ten rodzaj planety ma przypominać naszą Ziemię. Wyróżnione są morza i oceany (płaskie) oraz wystający ponad wodę ląd, pokolorowany podobnie jak na mapach - odpowiednie kolory przyporządkowano odpowiednim wysokościom nad poziomem morza.

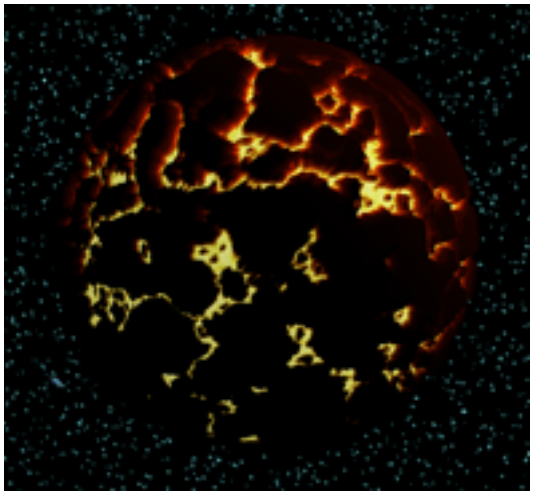
Aby osiągnąć ten efekt, szum Simplex poddawany jest następującym operacjom: dla każdego punktu mapy wysokości przypisywana jest wartość *max{0.48, stara_wysokość}*.



Planeta ziemio-podobna

5.1.2. Lawowa

Po całej powierzchni tej planety ciągną się rzeki roztopionej lawy, stały ląd zaś jest stanowią płyty zastygniętej magmy. Dodatkowo, lawa na tej planecie jest widoczna nawet po zacienionej stronie.



Planeta lawowa

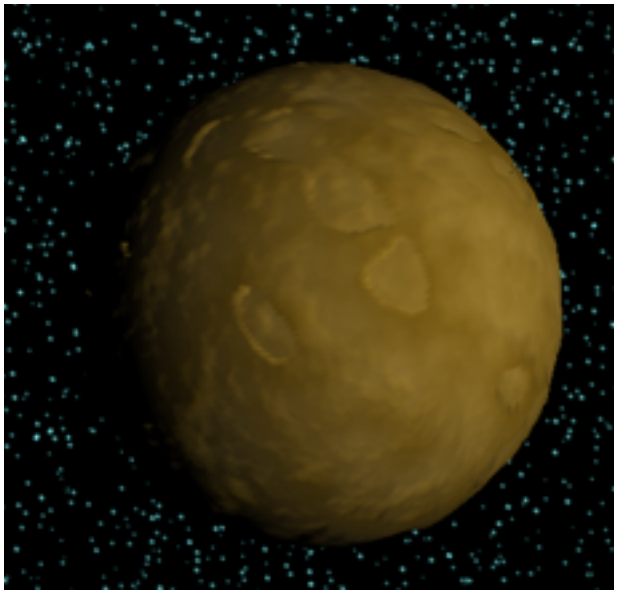
Aby osiągnąć efekt rzek, wykonano następujące działania: najpierw wartości poniżej 0.5 odwrócono w taki sposób, żeby znalazły się w przedziale [0.5; 1.0] wykonując operację *nowa_wysokość = 1.0 - stara_wartość* dla wszystkich punktów o wartości poniżej 0.5. Następnie wybrane wartości powyżej określonego progu zaokrąglono do 1.0, pozostałe zaś potraktowano funkcję wysokiego rzędu - w celu osiągnięcia stromych zboczy nad rzekami lawy.

Efekt świecącej lawy po ciemnej stronie planety uzyskano stosując specjalną teksturę do cieniowania, z której kolor pobierano na podstawie wysokości danego punktu na mapie wysokości.

5.1.3. Marso-podobna

Ta planeta ma burą, nieco rudawą powierzchnię. Jej charakterystyczną cechą są dużego rozmiaru kratery po uderzeniach komet.

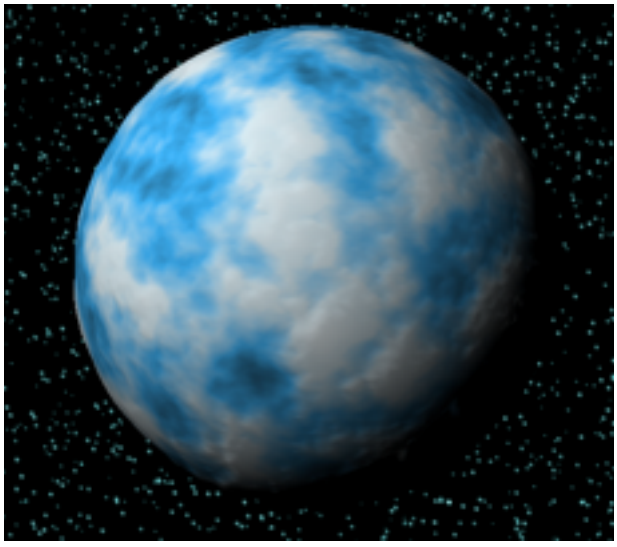
Aby osiągnąć ten efekt, dla oktawy o najniższej częstotliwości określono pewien próg, powyżej którego wykres funkcji szumu był odwracany wzdłuż osi przebiegającej na wysokości właśnie tego progu i punkt ten oznaczano jako krater, tak żeby kolejne warstwy szumu również były odwrócone.



Planeta marso-podobna

5.1.4. Lodowa

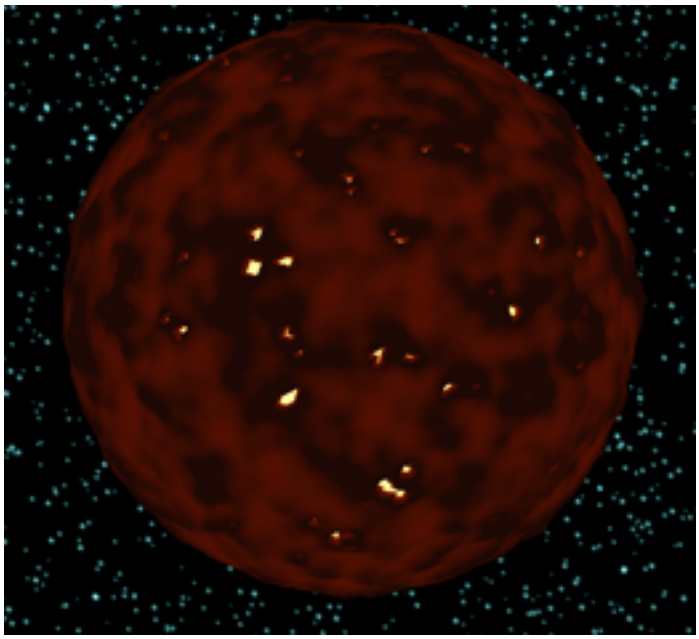
Ta planeta skuta jest grubą warstwą lodu pod którą znajduje się ciekły ocean. Tam gdzie lód jest cieńszy (wysokość na mapie wysokości jest niższa) tam ma on kolor niebieski, tak zaś gdzie lodu jest więcej, tam teren jest biały.



Planeta lodowa

5.1.5. Wulkaniczna

Planeta wulkaniczna pokryta jest licznymi kraterami wulkanicznymi. Efekt ten osiągnięto nieco podobnie jak kratery na planecie marso-podobnej, lecz tu wykorzystano wartości 2. i 3. oktawy, zaś wartość końcową (z zakresu [0.0; 1.0]) podniesiono do trzeciej potęgi, żeby uzyskać stromy zbocza wokół kraterów.



Planeta wulkaniczna

5.2 Układy gwiezdne

W centrum każdego układu umieszczana jest gwiazda - źródło światła. Następnie wokół tej gwiazdy rozmieszczane są w sposób pseudo-losowy kolejne planety. Działa to w taki sposób, że wyznaczany jest pewien pseudo-losowy promień, który w każdym kolejnym kroku jest zwiększany. Umowną sferę, jaką zakreśla wokół gwiazdy ów promień, można uznać za „orbitę" - pseudolosowe kąty *alfa* i *teta* określają punkt na tej sferze, w którym umieszczona zostanie planeta.

Wokół każdej gwiazdy można wyrenderować, bez wyraźnego spadku wydajności na komputerach spełniających minimalne wymagania, nawet do 100 planet.

Dzięki takiego algorytmowi, opartemu na wartościach pseudo-losowych, możemy zagwarantować, że dla danego ziarna (*seed*) otrzymamy zawsze ten sam układ gwiezdny. Ma to duże znaczenie przy generowaniu licznych układów i „podróżowaniu" pomiędzy nimi.

5.3 Kosmos

Przestrzeń kosmiczna w naszej aplikacji jest trójwymiarową kostką. Kostka ta jest podzielona w każdym wymiarze na n części, a zatem składa się z n^3 pomniejszych kostek. Każda jedna kostka to jeden układ gwiezdny.

Zawsze kiedy znajdujemy się w jednym z układów, niebo pokryte jest gwiazdami odpowiadającymi tym z innych układów. Kiedy kliknięciem myszą wybierzemy jedną z gwiazd na niebie, to zostaniemy do niej przeniesieni.

W efekcie po pierwsze niebo (to „na horyzoncie") z każdym miejscu jest dość podobne, ale nieidentyczne, a po drugie możliwe jest wrócenie do układu, który właśnie opuściliśmy, o ile uda nam się odszukać „naszą" gwiazdę wśród tysięcy innych.

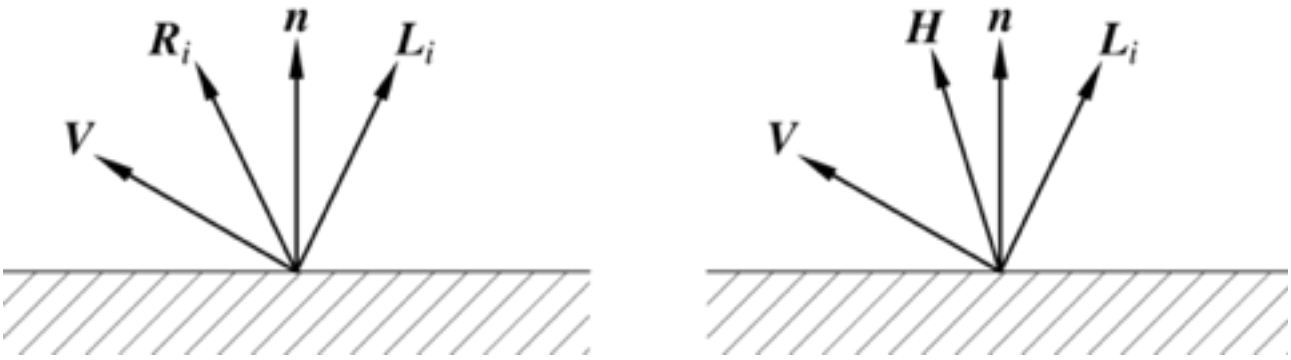
6. Iteracja nr 4

W tej iteracji celem było dodanie do aplikacji efektów graficznych związanych z oświetleniem i cieniowaniem. Wykorzystaliśmy do tego model oświetlenia Phonga.

6.2 Oświetlenie w modelu Phonga

Odblaski są możliwe do uzyskania w modelu opracowanym w 1975r. przez Phong Buoi-Tuonga. Wzór używany do obliczenia intensywności światła odbitego od powierzchni w kierunku obserwatora w tym modelu ma postać:

$$I = \left(I_{\mathrm{amb}} + \sum_i I_{\mathrm{d}i} v_i \max\left(0, \cos\measuredangle(\boldsymbol{n}, \boldsymbol{L}_i)\right)\right) a + \sum_i I_{\mathrm{d}i} v_i W\left(\measuredangle(\boldsymbol{n}, \boldsymbol{L}_i)\right) \cos^m \measuredangle(\boldsymbol{R}_i, \boldsymbol{v}).$$



Wektory występujące w modelu odbicia światła Phonga[1]

Oprócz światła otoczenia (*ambient*) oraz pewnej ilości punktowych źródeł światła, wzór ten posiada jeszcze składnik opisujący odblaski. Funkcja W uwzględnia zależność intensywności odblasku od kąta padania światła na powierzchnię i często przyjmowana jest tu funkcja stała. Intensywność światła w odblasku zależy od kąta między wektorem v opisującym kierunek do obserwatora i wektorem R_i, którego kierunek odpowiada odbiciu światła padającego z kierunku wektora L_i w idealnym lustrze.

[1] *źródło:* http://mst.mimuw.edu.pl/wyklady/gk1/phong.hd.png

Dzięki sprzętowej akceleracji, a także wsparciu ze strony frameworka THREE.js, zastosowanie metody Phonga okazało się być w granicach możliwości nawet przeciętnego komputera, którego parametry określono w ramach Analizy Wymagań.