TNM084
Generering av planeter i Unity

Malin Ejdbo

20 januari 2019

1 Introduktion och syfte

Användandet av procedurella metoder i datorgrafik är inte jättepopulärt och det är mer vanligt att använda texturer i spel och realtidsrendering. Ändå är procedurella metoder kraftfullt för att generera till synses slumpmässiga mönster som ger liv till scenen. Vid offline rendering är det vanligare att använda procedurella metoder eftersom det är förväntat att renderingen tar längre tid. Denna rapport presenterar ett projekt som har använt olika procedurella metoder för att generera en planet som användaren kan kontrollera utseendet på.

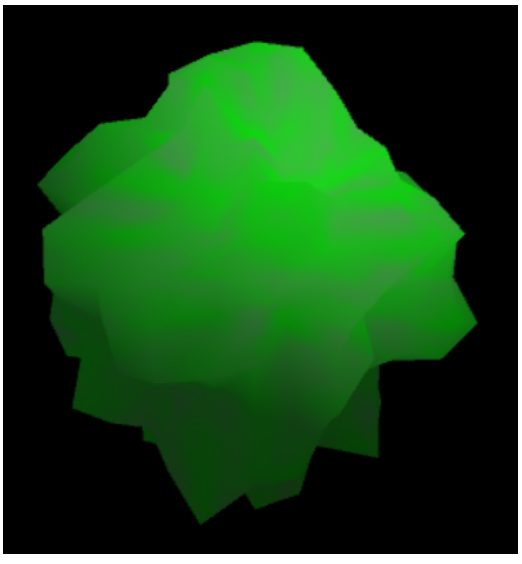
Målet med projektet är att procedurellt skapa en planet som användaren kan manipulera. Planeten är i skala ungefär lika stor som en snöboll. Användaren ska ha möjlighet att göra planeten till sin egen och kunna sätta sin egen prägel på den inom rimliga gränser. Det ska inte vara möjligt för användaren att sätta sådana extrema parametrar att resultatet blir alltför galet och obrukbart.

2 Metod

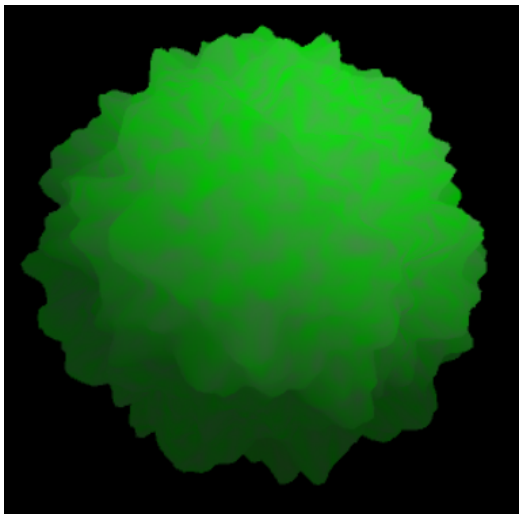
Lite teori om de metoder som användes vid implementeringen av projektet nämns i detta avsnitt.

2.1 Planetens bas

En planet är i det stora hela sfärisk och därför användes en sfär som grund för planeten. För att skapa terrängen på ytan av sfären manipulerades den på vertex-nivå. För att göra detta snyggt krävs det att sfären har en tillräcklig hög upplösning av verticer. Detta krävs för att kunna manipulera sfären med mjuka kanter som kan ses i figur 1. På liknande sätt som att en lågupplöst bild ser suddig ut och kan missa viktiga detaljer behövs en högupplöst modell för att kunna återge alla detaljer på ytan.



(a) Sfär med lågt antal verticer. Kanterna blir vassa.

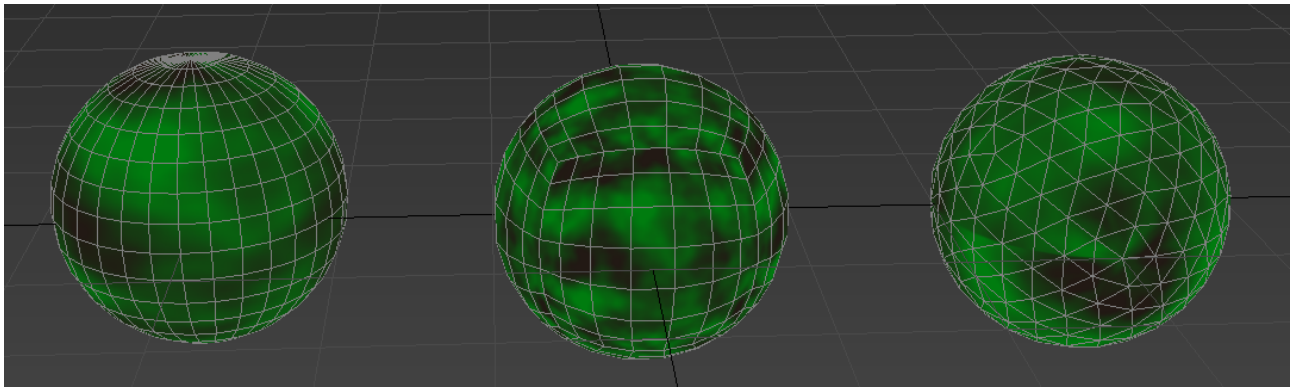


(b) Sfär med många verticer. Kanterna blir mjukare och mer detaljer blir synliga.

Figur 1: Jämförelse mellan två sfärer med olika antal verticer. Båda sfärerna har manipulerats med samma parametrar.

Ett annat problem med en sfär är hur trianglarna fördelas över ytan. Det vanligast sättet är att sfären byggs upp med sfäriska koordinater, detta ger en ojämn fördelning av trianglarna då det blir fler vid

polerna än vid ekvatorn på sfären [1]. Trianglarnas storlek varierar också då de vid polerna är väldigt små och de vid ekvatorn blir stora. Det optimala är när trianglarna är jämnt fördelade på ytan med ungefär samma storlek. Figur 2 visar en jämförelse mellan tre sfärer som är uppbyggda på tre olika sätt.



Figur 2: Tre sfärer uppbyggda med tre olika metoder. Sfären till vänster skapades med sfäriska koordinater, den i mitten är en *sfärifierad* kub och den till höger är en *geosfär*.

Sfären i mitten är en kub som har *sfärifierats* [2], alltså manipulerats till att se ut som en sfär. Den sfär som är till höger är en *geosfär* [3], alltså en sfär som i basen består av en *ikosaeder* [4]. Som kan ses i figur 2 har geosfären en pol på framsidan vid ekvatorn där mönstret knyter ihop till en punkt. På grund av detta valdes den sfäriferade kuben som grund för planeten.

2.2 Noisefunktioner

För att planeten inte ska se plastig ut och ha ett mer slumpmässigt mönster användes olika noisefunktioner. De noisefunktioner som användes var *simplex noise* [5] och *cellular noise* [6], båda i tre dimensioner. noisefunktionerna användes för att skapa en slumpmässig terräng och för att ge mer liv i färgerna. Implementationen av noisefunktionerna är skriven av *Stefan Gustavson* och hämtades från GitHub[1].

3 Implementation

Projektet implementerades i Unity i kodspråket *C#* och *HLSL* som shading språk. Detta innebar att noise implementationerna[1] översattes från *GLSL* till HLSL.

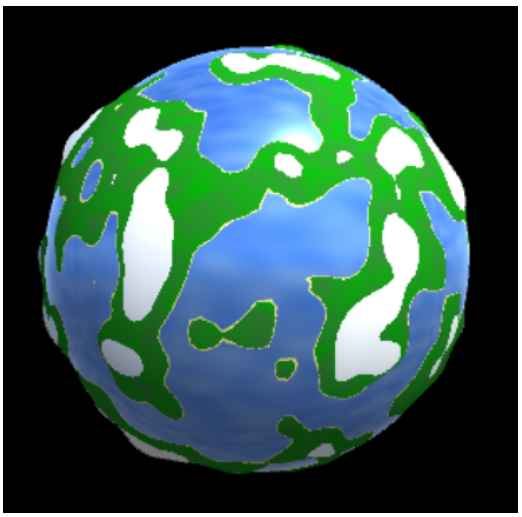
3.1 Terräng

Varje planet behöver en terräng som utgör ytan på planeten. Denna ska kännas slumpmässig och både ha berg och dalar spridda över ytan. Detta implementerades med en vertex shader som flyttar varje vertex i sfären lite ut eller in från sfären. För att göra denna process slumpmässig användes simplex noise i tre dimensioner. För att öka känslan av att det är slumpmässigt och inte jämnt fördelat över ytan användes två fraktaler av simplex noise. Alltså orginalet plus samma noise fast med dubbla frekvensen och halverad amplitud. Amplituden halveras på grund av att det mer högfrekventa mönstret annars skulle ta över.

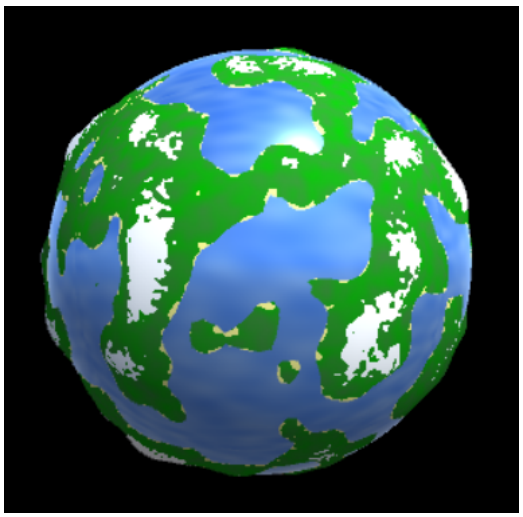
[1] Noise implementationer: `https://github.com/stegu/webgl-noise`

3.2 Färg

Vid färgsättningen av planeten var tanken att terrängens färg skulle bero på hur hög terrängen är. Alltså att toppen av höga berg är vitare för att likna snö och kanterna runt vattnet är gulare för att likna stränder. Användaren kan sedan välja vilka färger som ska representera snö och stränder. För att besluta vilken färg som ska appliceras i *surface shadern* [7] beräknas avståndet till centrum av sfären, detta är för tillfället hårdkodat men borde bero på vart sfären ligger i scenen. Om avståndet är större än en höjd som bestäms vara bergshöjden eller mindre än höjd som bestäms vara strandhöjden, blandas snö eller strandfärgen ihop med planetens basfärg. Detta gav ganska distinkta områden för snö och strand som såg onaturligt ut som kan ses i figur 3a. För att undvika detta lades lite simplex noise till på avståndsberäkningen, detta gav mer slumpmässiga områden för snö och stränder och såg mer naturligt ut som kan ses i figur 3b. I figur 3 kan ses en jämförelse innan och efter simplex noise lades till i avståndsfunktionen.



(a) Resultat när snö och strand lades till beroende på avståndet till planetens centrum.



(b) Resultat när slumpmässighet lades till i beräkningen av avståndet till planetens centrum.

Figur 3: Jämförelse mellan resultatet med eller utan att lägga till lite slump vid beräkningen av avståndet till planetens centrum.

För att få mer liv i planetens basfärg användes simplex noise även för att blanda två olika färger som planetens basfärger. Värdet från simplex noise bestämde hur mycket dem två olika färgerna ska blandas. Detta kan användaren styra genom en variabel som styr hur mycket av den sekundära färgen som blandas in i den primära färgen. Ju högre detta tal är, desto mer av den sekundära färgen träder fram och även kontrasten mellan båda färgerna ökar. Terrängens material sattes även till att vara diffust för att inte reflektera solens ljus i ytan.

3.3 Vatten

För att skapa hav och sjöar på planeten användes ytterligare en sfär som är lite mindre än sfären för planeten. Detta innebär att vissa av planetens dalar blir vattenfyllda och skapar hav och sjöar. Denna sfär har inte manipulerats på vertex-nivå men har en annorlunda färgsättning än terrängen. För att skapa vågor i vattnet användes simplex noise i ett mer avlångt mönster för att likna vågor. Två färger blandades, vattnets basfärg och vågornas färg. Vattnets material har även en spekulär del för att likna att solens ljus reflekterar i vattenytan.

3.4 Stjärna

En stjärna lades till i scenen för att öka antalet objekt som användaren kan manipulera. Tanken i början var att planeten ligger i omloppsbana runt stjärnan och användaren följer med planeten i omloppsbana. Detta blev problematisk dock då användaren även skulle kunna röra sig i scenen fritt. Därför beslöts det att istället för att planeten är i omloppsbana runt stjärnan, är stjärnan i omloppsbana runt planeten. Detta för att underlätta för användaren att navigera i scenen.

Utseendet på stjärnan är lite annorlunda än utseendet på terrängen. Cellular noise användes för att färgsätta stjärnan. Denna noisefunktion passar bättra på stjärnan än simplex noise på grund av att prickarna i mönstret liknar solfläckar. Användaren kan sedan välja vilka två färger som ska blandas i mönstret. Stjärnans material sattes även till att vara emissivt och sänder ut ljus i scenen. Det ljus som ljussätter planeten är dock inte detta emisiva ljus. Inuti sfären för stjärnan är en punkt-ljuskälla placerad som står för ljussättningen av planeten som är relativt långt ifrån stjärnan. Det emisiva ljuset används endast för att belysa stjärnan själv. Eftersom ljuskällan är inuti stjärnan når inte dess ljus utsidan av stjärnan, detta gör att den ser ganska mörk ut utan emisivt ljus.

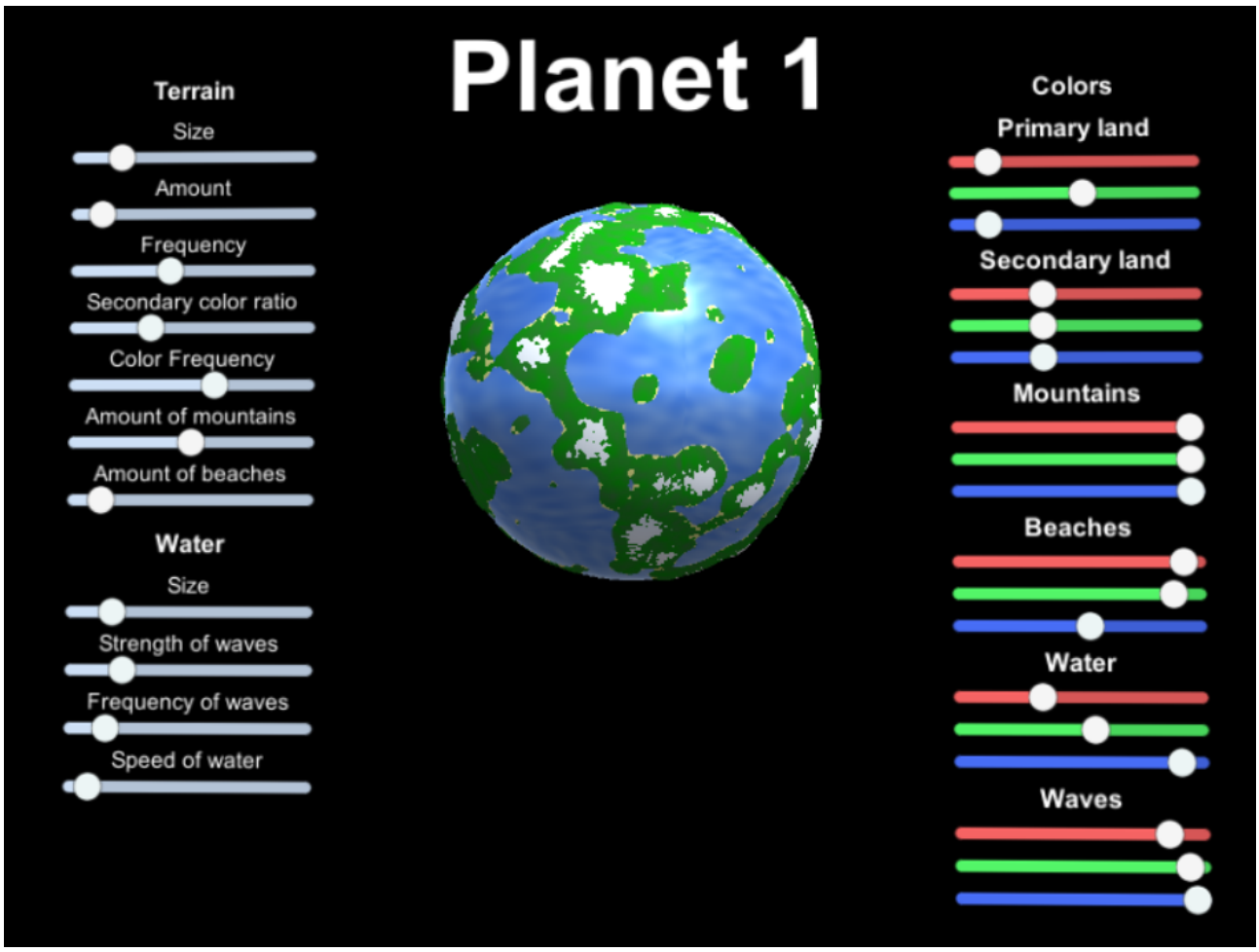
3.5 Animationer

För att ge scenen mer liv lades några animationer till. Stjärnann rör sig i en omloppsbana runt planeten och både stjärnan och planeten snurrar runt sin egen axel. Vattnet på planeten animerades även genom att använda en tre dimensionell metod för simplex noise där en koordinat beror på tiden. På liknande sätt animerades även ytan på stjärnan med cellular noise i tre dimensioner.

Användaren kan även navigera runt i scenen med hjälp av ett skript. Navigationen liknar det som finns i första persons spel där man även kan flyga.

3.6 Gränssnitt

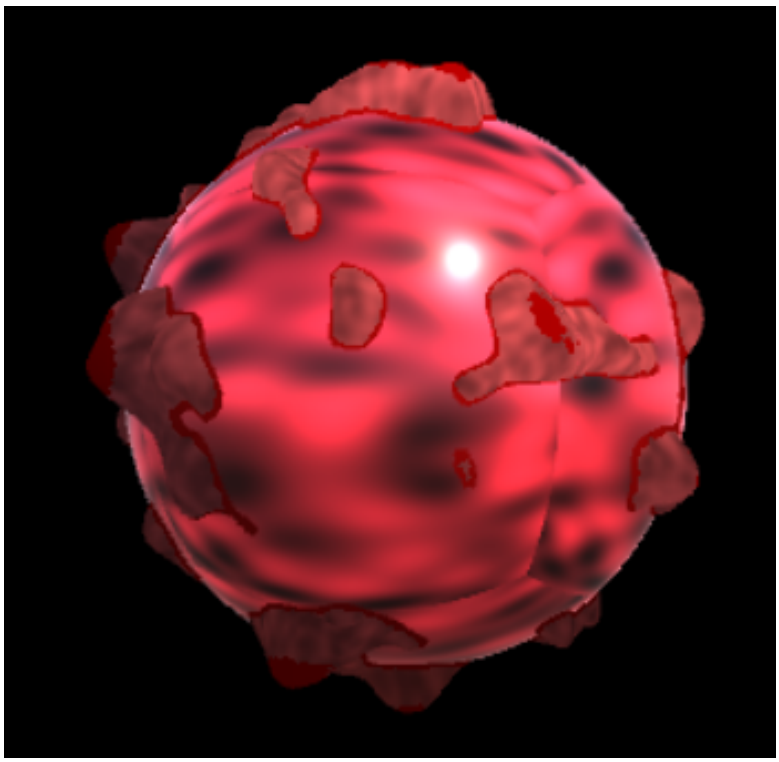
Målet med projektet är att användaren ska kunna manipulera planeten och stjärnan som den önskar. För att kunna göra det måste det finnas något användargränssnitt som användaren kan integrera med som påverkar utseendet på objekten. Ett ganska enkelt gränssnitt implementerades så användaren kunde dra i olika reglage för att påverka objekten. I figur 4 ses en bild på det slutgiltiga gränssnittet. Gränssnittet byggdes upp av Unitys inbyggda standard objekt för grafiskt gränssnitt där endast färgen ändrades.



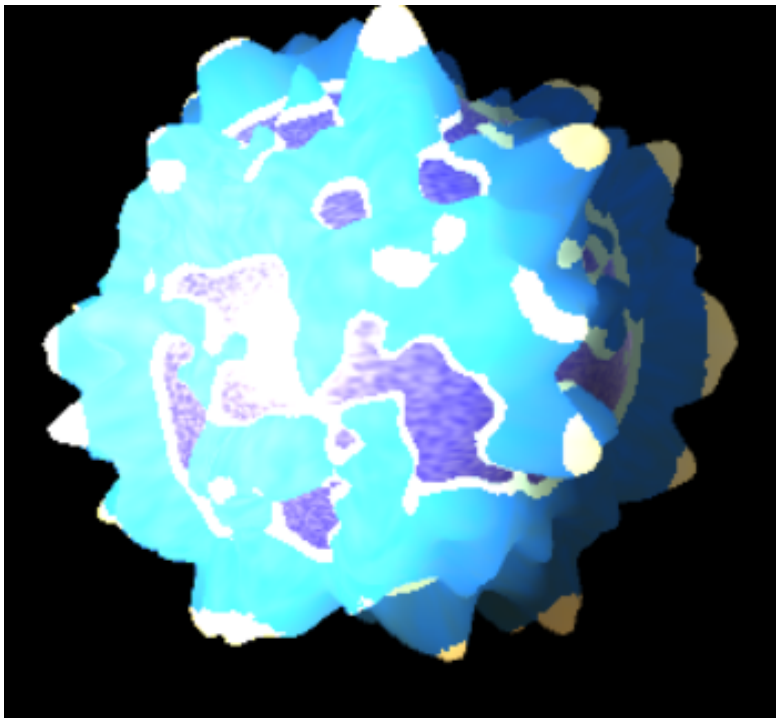
Figur 4: Det grafiska gränssnitt som användaren använder för att manipulera planeten.

4 Resultat

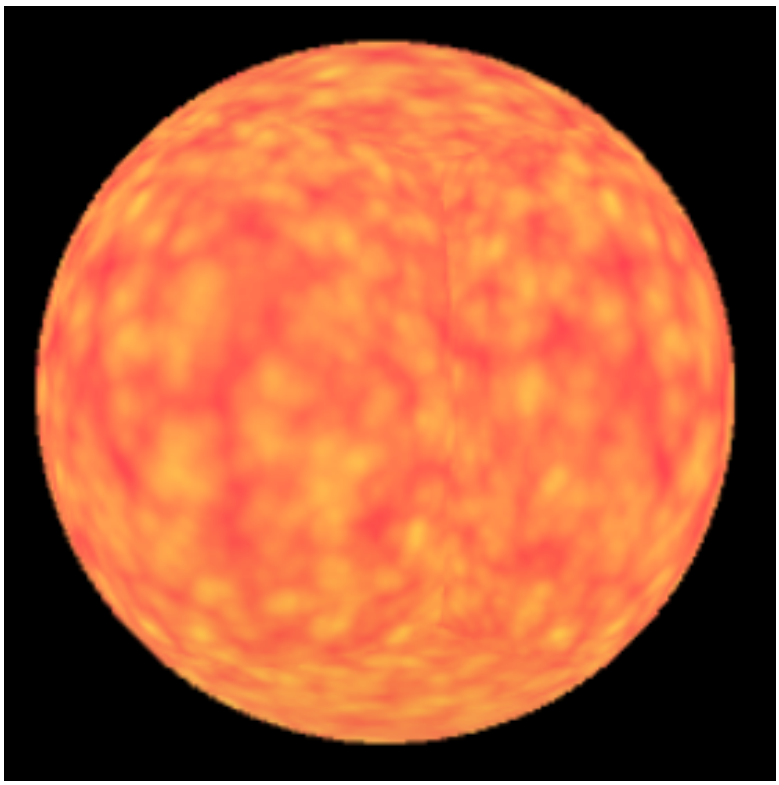
Resultatet av projektet är en scen med en planet och en stjärna som användaren kan manipulera med ett enkelt grafiskt gränssnitt. Några exempel på olika planeter som användaren kan tänkas skapa kan ses i figur 5, 6 och även som tidigare setts i figur 3b. Ett exempel på en stjärna kan ses i figur 7.



Figur 5: En planet täckt av lava och vulkaner.



Figur 6: En planet som är väldigt bergig och täckt av is och snö.



Figur 7: En stjärna som liknar solen i vårt eget solsystem.

5 Diskussion

Något som skulle kunna förbättras för att få scenen lite snyggare är bland annat att ha en mer jämn övergång i färgen från strand till terräng och från terräng till berg. Just nu är det en ganska hård övergång från en färg till en annan. Det skulle även ge mer liv om strandfärgen och snöfärgen inte var homogen utan även den innehöll någon noisefunktion för att inte se plastig ut. Eller om några andra noisefunktioner hade används, till exempel *flow noise* [8] för vattnet.

En annan sak som skulle kunna förbättras är realismen i scenen. Nästan ingenting är baserat på fysikalisk data. Skalan på objekten är fel då en stjärna är mycket större än en planet och avstånden mellan himlakroppar är väldigt stora. En annan skala som är fel är hur mycket av terrängens variation som egentligen är synlig från rymden. Som det ser ut nu så går det att få väldigt tydliga berg på planeten som inte är realistiskt. En förbättring vore att förhindra det och hålla parametrarna inom rimlig skala. Men på en liten modellplanet så skulle det innebära att det i princip är en sfär med väldigt små ojämnheter och det är inte lika roligt att titta på som en väldigt bucklig planet med mycket berg.

Några idéer om hur projektet skulle kunna skalas upp är att bland annat skala upp planeten i sig. Detta innebär givetvis andra utmaningar som precisionsproblem och *level of detail* kommer krävas för att kunna hantera alla trianglar i scenen då upplösningen på sfären kommer behövas vara mycket högre. Detta skulle kunna ge planeten mer realism då användaren skulle kunna gå på planetens yta och se hur stora bergen och dalarna ser ut från ytan och inte från rymden. När användaren sedan har skapat sin modellplanet skulle denne kunna spara denna i ett galleri som kan samla på flera olika planeter. Dessa planeter skulle sedan kunna placeras ut i ett solsystem i omloppsbana runt en stjärna som även den går att manipulera som användaren vill. Omloppsbanan skulle kunna räknas ut matematiskt med gravitation och planeternas massor för att öka realismen. Användaren kan nu skapa planeter och stjärnor, då sakans bara att den även kan skapa månar. Dessa månar skulle då åka i omloppsbana runt dess planet.

Referenser

[1] Sebastian Lague. *[Unity] Procedural Planets (E01 the sphere)* [Video] 2018, 16 Aug [citerad 2019-01-19]. Hämtad från:
https://www.youtube.com/watch?v=QN39W020LqU&t=162s

[2] Autodesk.Help. *Spherify Modifier* [Internet]. Autodesk Inc; 2017 [uppdaterad: 2017-06-15; citerad: 2019-01-19]. Hämtad från:
https://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2017/ENU/3DSMax/files/GUID-3BC50BC2-97DA-4271-AC00-C46DA6C71FE4-htm.html

[3] Autodesk.Help. *GeoSphere* [Internet]. Autodesk Inc; 2016 [uppdaterad: 2016-02-08; citerad: 2019-01-19]. Hämtad från:
https://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2016/ENU/3DSMax/files/GUID-ADCB4D06-BD52-4D0A-A367-63FCBC37A848-htm.html

[4] Nationalencyklopedin [Internet]. NE Nationalencyklopedin AB; 1985-. *ikosaeder* [citerad: 2019-01-19]. Hämtad från:
https://www-ne-se.e.bibl.liu.se/uppslagsverk/encyklopedi/l%C3%A5ng/ikosaeder

[5] Gustavson S. (stegu@itn.liu.se). *Simplex noise demystified* [Internet]. Sweden: Linköping University; 2005 [uppdaterad: 2005-03-22; citerad: 2019-01-20]. Hämtad från:
http://weber.itn.liu.se/~stegu/TNM084-2017/simplexnoise.pdf

[6] Worley S. *A Cellular Texture Basis Function* [Internet]. 1996 [citerad: 2019-01-20]. Hämtad från:
http://weber.itn.liu.se/~stegu/TNM084-2017/worley-originalpaper.pdf

[7] Unity Technologies. *Writing Surface Shaders* [Internet]. Unity Technologies; 2017 [uppdaterad: 2017-06-08; citerad: 2019-01-20]. Hämtad från:
https://docs.unity3d.com/Manual/SL-SurfaceShaders.html

[8] Perlin K. *Flow Noise* [Internet]. Okänt år. Hämtad från:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.141.5266&rep=rep1&type=pdf