Eldboll

TNM084 Procedurella bilder

Hannes Ingelhag

1 Introduktion

Internet växer fortfarande så det knakar och 3D grafiken har även letat sig dit. WebGL gör det möjligt för vem som helst som har en webbläsare att enkelt komma igång och programmera 3D-grafik som beräknas på datorns GPU. I denna rapport kommer det beskrivas hur det är möjligt att implementera en verklighetstrogen eldboll med hjälp av procedurella metoder. Till hjälp kommer WebGL att användas tillsammans med programmeringsspråket GLSL för att skriva shaders. De två noisefunktionerna som har används för att skapa de olika elementen i eldbollen är Simplex- och Worley-noise.

2 Teori

Det är möjligt att använda noisefunktioner för att skapa visuella mönster i datagrafik. Eftersom inga material är helt utan textur är det möjligt att genom att addera slumpartade artefakter få det att se verkligt ut. Genom att justera in-parametrarna på funktionerna är det möjligt att styra slumpen och på så sätt efterlikna det eftersträvade materialet.

Den mest kända noisefunktionen är Perlin noise[1], även kallat klassiskt noise. Perlin noise beror på en falsk slumpmässig gradient som är sammankopplad med de närmaste rutnätspunkterna. Funktionen beror huvudsakligen på interpolation till den närmaste punkten i rutnätet tillsamans med gradienten [2]. Dock är denna funktion ganska beräkningstung då den alltid förhåller sig till fyra punkter i rutnätet. Därför utvecklades en annan noisefunktion - Simplex noise. Istället för en kvadrat, som Perlin noise, använder denna funktion en triangel för att beräkna intensiteten i den nuvarande punkten beroende på de tre närliggande punkterna i triangeln.

Worley noise[1] används ofta för att simulera texturer för vatten, stenar eller andra cell-liknande material. Funktionen är baserad på att ett antal slumpmässiga egenskapspunkter placeras ut med en viss variation i rymden. Sen är det möjligt att skicka tillbaka den närmsta eller de två närmsta egenskapspunkterna. I detta projektet användes de två närmsta punkterna då det ger en mer variation till texturen. Detta är en relativt långsam funktion jämnfört med Perlin noise då den måste sortera alla egenskapspunkter beroende på avståndet mot den aktuella.

- *Hannes Ingelhag, Linköping University, Sweden*
  e-mail: hanin602@student.liu.se

3 Implementation

Projektet är programmerat i WebGL tillsammans med GLSL som de olika shaders är skrivna i. För att skapa enkla geometrier och för att enkelt komma igång med projeketet användes javascriptbiblioteket Three.js.

Eldbollen, som är gjord i 3D, är uppbyggd av tre olika element; rök, eld och den inre bollen. Dessa element är uppbyggda av två olika sorters geometrier, sfärer och cylindrar.

3.1 Noise

För att skapa eld och rök används simplex noise både i vertex- och fragmentshadern. I alla fall används elva oktaver av simplex noise för att skapa en realistisk simulation, *se ekvation 1*. Den inre bollen använder sig av Worley noise för att få en cell-liknande textur där det ser ut som att materialet har krackelerat av värmen. Mer beskrivet hur och när noiset används beskrivs under respektive underrubrik.

$$\sum_{n=0}^{10}(1/2^n) * abs(snoise(vec3(2^n * xPos, 2^n * yPos, 2^n * zPos))) \tag{1}$$

3.2 Färg

När noise-funktionerna används retuneras ett skalärvärde mellan -1 och 1. För att mappa värdet till en färg som ska efterlikna eld används *kodstycke 1*. Detta används både för den inre bollen och eldsfärerna som cirkulerar runt om den inre bollen och det görs i fragment-shadern.

Kodstycke 1: Funktionen mappar om ett skalärvärde till ett rgb-värde

```
vec3 toFireColor(float n) {
    float red = clamp(3.0*n,0.0,1.0);
    float grn = clamp(3.0*n-1.0,0.0,1.0);
    float blu = clamp(3.0*n-2.0,0.0,1.0);

    return vec3(red,grn,blu);
}
```

3.3 Inre bollen

Den inre bollen använder sig enbart av en fragmentshader för att skapa en krackelerad textur, i övrigt är bollen en perfekt sfär. Texturen skapas genom *kodstycke 2* där *amplituden* beskriver hur stora sprickor det kommer att bli och där *time* beskriver hur texturen kommer att förändras beroende på tiden. I det här fallet går det att utläsa att det blir många och små sprickor och att det

kommer att se ut som att sprickorna rör sig uppåt i y-led och bakåt i z-led.

Kodstycke 2: Hur texturen räknas ut för den inre bollen med hjälp av Worley noise

```
float amplitude = 3.0;
vec2 F = cellular(
    vec3(   xPos*amplitude,
            yPos*amplitude - time*0.2,
            zPos*amplitude - time));
float intensity = F.y - F.x;
```

3.4 Eld

Från början är elden flera genomskinliga sfärer som ligger runt den inre bollen med varierande radie. Dessa sfärer deformeras i vertex-shadern i följande steg:

1. Positionerna multipliceras med y-värdet så det blir en ellips
2. X- och Z-positionerna multipliceras med en smoothstep-funktion som beror på y-värdet så att elden dras samman längre upp.
3. Simplex noise som beror på x,y,z-position och tiden appliceras och gör så att sfären vobblar fram och tillbaka.
4. Ju längre upp i y position sfären är desto mer flyttas x- och z- positionerna för att simulera att flammorna fladdrar i vinden.

För att simulera eldflammor på de nu skapade sfärerna används Simplex noise i fragment-shadern med *ekvation 2*. Enligt ekvationen ändras amplituden på y-positionen linjärt medan x- och z-positionen ändras expontiellt, detta för att få långa och avsmalande flammor.

$$\sum_{n=0}^{10}(1/2^n) * abs(snoise(vec4(\quad 2^n * xPos, \; 0.25*n*yPos - 1.0*n*t, \; 2^n * zPos, \; 0.5 + 0.1*n))) \tag{2}$$

3.5 Rök

Skapa rök som ser bra ut från olika håll är svårt att göra. Efter många olika försök implementerades flera lager med cylindrar som startar i mitten på den inre sfären och sträcker sig upp i y-led. Dess vertex-shader ändrar x- och z- position beroende på en noisefunktion som är skalad på y-värdet på samma sätt som elden som står beskrivet ovan. I fragment-shader används simplex noise för att skapa ett rökliknande skal.

4 Resultat

I den slutgiltiga demon är det möjligt för användaren att ställa in kvalitén på röken och eldens. På elden ökas antalet sfärer och antal polygoner och för röken adderas endast flera lager. *Figurerna 1-6* visar olika resultat med olika sorters kvalité på röken och elden. Bästa sättet att se eldbollen är i realtid på en dator med kraftfullt grafikkort.

5 Diskussion

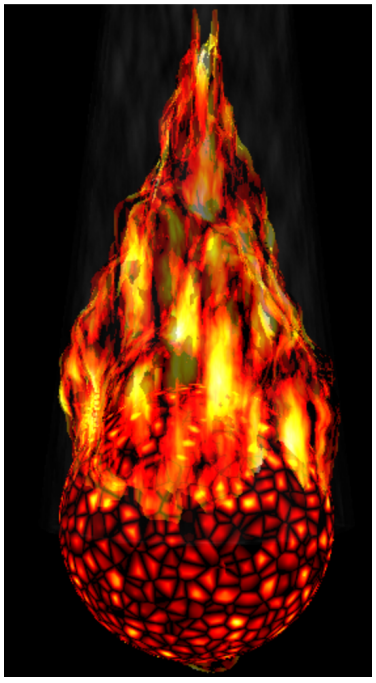
Procedurella metoder för att skapa simuleringar av verkliga artefakter är mycket användbart och att samtidigt kunna göra det i realtid blir mer och mer realistisk med hjälp av dagens grafikkort. Nu när WebGL blir bättre utvecklat blir det ännu med intressant då det går att använda mobila enheter där beräkningskraften finns och minnesstorlekarna saknas, för att spara stora och högupplösta texturer.
Från början var tanken att skapa en eldboll där det var möjligt att användaren kunde interagera med eldbollen och flytta runt den i rymden för att skapa olika sorters mönster med rök och eld. Efter många olika försök fick detta däremot åsidosättas då röken var för svår att få bra i en 3D-rymd när vind skulle adderas. Slutresultatet blev då en sfär som roteras i y-led och där eldflammorna fladdrar med olika hastighter.

Jag är nöjd med den inre bollen och elden, men röken som jag har arbetat med mer än 60% av tiden är fortfarande inte riktigt där jag vill ha den. Det hade varit enklare att göra hela projektet i 2D istället.

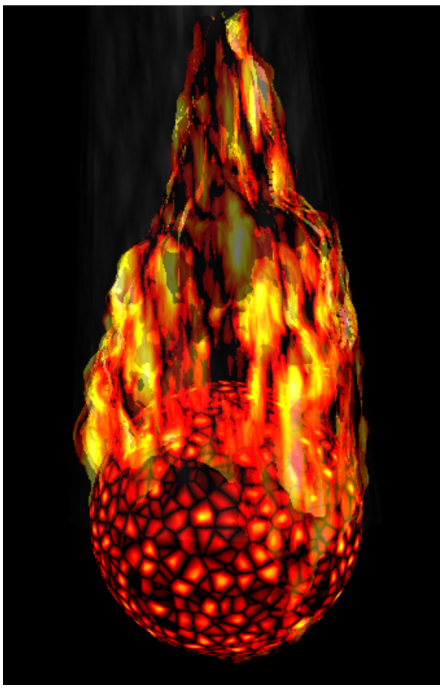
Referenser

[1] Patrick Cozzi and Christopher Ricco, 2012/05/08, OpenGL Insights, ISBN 978-1-4398-9376-0, Baco Raton, CRC Press

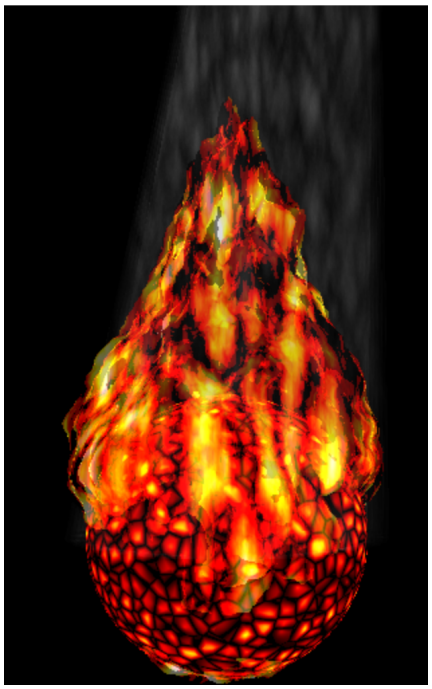
[2] Stefan Gustavson, 2005, Simplex noise demystified, http://webstaff.itn.liu.se/ stegu/ TNM084-2014/simplexnoise.pdf



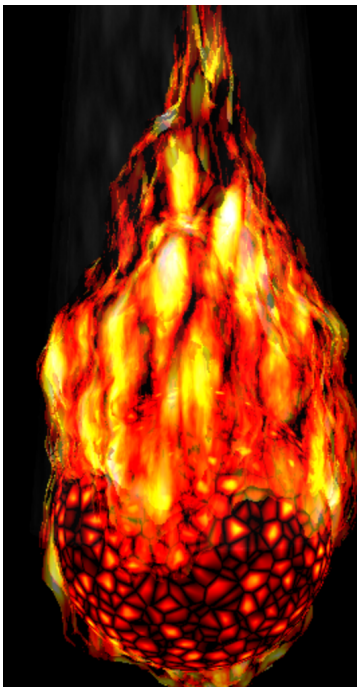
Figur 1: Kvaliteten på rök och eld är på lägsta



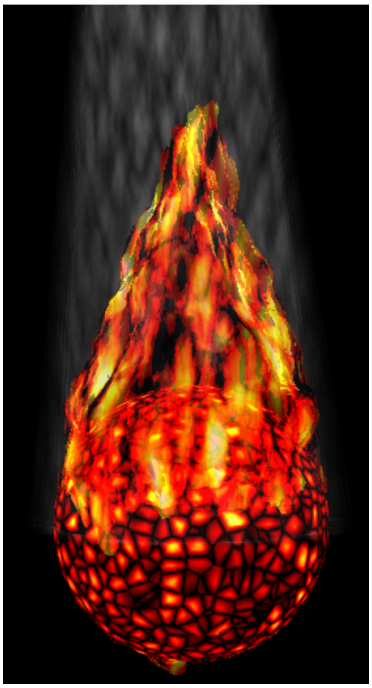
Figur 2: Kvaliteten på rök är på lägsta och eld medium



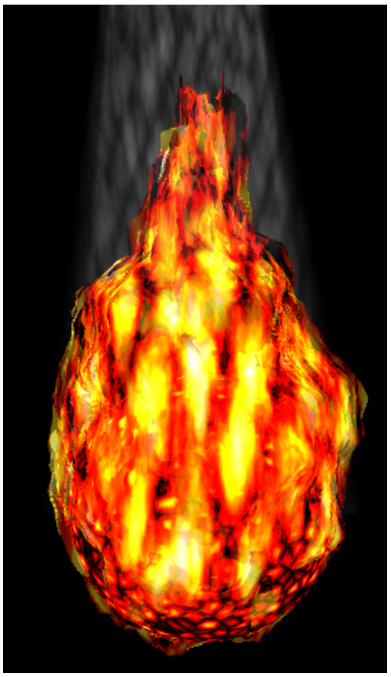
Figur 4: Kvaliteten på rök är medel och eld är på lägsta



Figur 3: Kvaliteten på rök är på lägsta och eld högsta



Figur 5: Kvaliteten på rök är högsta och eld är på lägsta



Figur 6: Kvaliteten på rök och eld är på högsta