BACHELOR THESIS

Animated Foliage and Cloth for Mobile Games

Linus Lindberg
2016

Bachelor of Fine Arts
Computer Graphic Arts

Luleå University of Technology
Department of Arts, Communication and Education

LULEÅ UNIVERSITY OF TECHNOLOGY

BACHELOR THESIS

Animated Foliage and Cloth for Mobile Games

Linus Lindberg

Abstract

Problem Summary

Realistic cloth and foliage effects can be very calculation intensive, and therefore they are not very common in mobile games. Using effects that are both visually appealing and light on performance could greatly improve the look of mobile games that usually don't have effects like these. I chose to limit the scope of the project to these cases.

- Cloth or foliage blowing in the wind
- Foliage reacting to touch, like a player walking through it
- Cloth or foliage reacting to bullets

Is it possible to, instead of using expensive physics calculations to simulate these effects, just approximate the 'look' of it but still base it on reality?

Result

It is helpful to start with a look at physics even when writing simplified, non-physical effects. By knowing exactly what causes the look you are after it's much easier to mimic. I discovered that material stiffness is the most important property to define how a branch, some grass or some hanging cloth reacts to external forces.

I decided to go with a shader based solution for all three effects, because of the performance benefits over joints and vertex simulations. The wind shader is a common technique using sine waves to make an offset to the vertex positions before drawing to the screen. The shader for foliage reacting to touch builds on this by supplying it with a player position and movement velocity. This is then used to make additional changes depending on how far away the player is.

The shader reacting to bullets is very similar, but instead of player position it is supplied with bullet positions along the shooting vector. The vertices are the displaced by how close they are to this position creating an effect of the cloth stretching and then falling back in the direction of the shot.

The first two effects worked out well, with good performance and a natural look. The shooting effect though needs further improvement to be used in a game. Overall I believe shader tricks like these can be very useful in mobile games that want some extra detail but can't afford vertex simulation.

Sammanfattning

Problem

Realistiska tyg- och löveffekter kräver mycket processorkraft, och är därför inte särskilt vanligt i mobilspel. Att använda effekter som både är billiga och snygga skulle höja den visuella kvalitén i spel som idag inte har några sådana effekter. Jag fokuserar i den här rapporten på att lösa följande situationer:

- Tyg eller växtlighet som blåser i vinden
- Växt som reagerar när spelaren springer igenom den
- Växt eller tyg som reagerar på skott

Är det möjligt att istället för att använda dyra fysiska algoritmer, bara skapa en effekt som försöker efterlikna dessa?

Resultat

Det underlättar att först lära sig hur dessa effekter fungerar i verkligheten, även om man gör en förenklad effekt som inte direkt använder sådan information. Genom att veta vad som bidrar till effekten du är ute efter är det mycket lättare att försöka återskapa den. Under arbetet upptäckte jag att ett materials styvhet är den viktigaste variabeln för att definiera hur en gren, gräs eller hängande tyg reagerar med sin omgivning.

Jag valde att lösa alla tre situationerna med shaders, eftersom de är snabbare än animerade joints och vertexsimulationer. Vind-shadern är en sinusvåg som lägger till ett värde på vertexen innan den ritas på skärmen, en vanlig lösning i spel. Shadern för växten som reagerar när spelaren går igenom den fortsätter bygga på detta genom att räkna in spelarens position och riktningen den rör sig i, för att göra så att växten rör på sig och böjer sig när spelaren kommer nära.

Shadern för materialet som reagerar på skott är väldigt likartad, men istället för att läsa spelarens position får den kulans position och riktning. Vertex-punkterna flyttas med riktningsvektorn baserat på hur nära kulan är.

De första två effekterna fungerar bra, med bra prestanda och naturlig rörelse. Skotteffekten behöver dock mer arbete för att kunna användas i ett riktigt spel. Jag tror att shadereffekter som de här är väldigt användbara i mobilspel som vill ha lite extra visuella detaljer, utan vertexsimulation.

Table of Content

Innehåll

Glossary

Shader
A shader is a set of instructions that describes how an object should be drawn on the screen

Vertex Program
The vertex program is the most important part of a shader, it's primary job is to take the input positions from object space and convert them into screen space so that they can be drawn on the screen. Cheaper shaders do a lot of lighting calculations here before sending it to the fragment program.

Fragment Program
The fragment program draws the object on the screen. More advanced shaders do their lighting here, on a per fragment basis for a smoother more accurate light.

Overdraw
Overdraw is when you draw a pixel more than once due to alpha.

CPU
CPU stands for central processing unit. It does most of the calculations not related to graphics.

GPU
The GPU, or graphics processing unit, is dedicated to doing calculation related to graphics, like drawing objects on the screen

1 Introduction

Nothing breaks the immersion of a game like completely static foliage and cloth. These are light materials that are largely defined by how they react to outside factors. Imagine a flag that doesn't flow with the wind. It looks weird and doesn't read like cloth to an observer.
Modern console and PC games have started using real time physics simulation to get very convincing reactions of these materials. For low-end computers and mobile devices this is simply not an option due to the sheer amount of calculations required for such an effect.
This report will explore different ways of faking these properties, starting with a quick look at how these materials react in real life and then studying how they can be approximated with a heavy focus on performance.

1.1 Background

I wrote this thesis while on an internship at Level Eight, a mobile game company in Umeå Sweden. When they heard I was interested in technical art they asked me if I could explore a solution of making vegetation and cloth more interesting, as they were completely static. One crucial part was of course performance, after all this was for a mobile game. After looking into this I discovered that the subject is not very well documented. Today the focus of studies like this lies exclusively on simulations of cloth and foliage which are impractical for anything but the most powerful devices.

There is obviously a need for simplified solutions. This is what I want to look into in this report.

1.2 Purpose

The purpose of this publication is not to find new ways to accurately simulate movement of cloth in real time but instead to use existing methods to create movement that looks plausible. The goal is to fool the casual observer that there are physical effects taking place, but not actually performing complex algorithms needed to simulate physically correct movement.

In mobile games physical realism needs to take a back seat to performance. I will need to figure out a method that both looks good and works well on these devices.

The key is that the method should look realistic, but in the simplest way possible.

1.3 Problem

The focus of this report is discovering how animated cloth and foliage effects that work well on mobile devices can be created.

To solve this problem these following questions first need to be answered:

- **What properties makes cloth and foliage *appear* like cloth and foliage? What are the physics behind this in real life?**

- **What methods are commonly used to simulate these properties in computer graphics?**
- **What methods works best for mobile devices and why?**

1.4 Limitations

To limit the scope of the report, the problems will focus on the following cases:

- **Hanging cloth on static object, or vegetation reacting to wind**
- **Vegetation, reacting to touch and wind**
- **Hanging cloth reacting to bullets**

2 Theory

2.1 Examining the real world

What makes cloth move different from a sheet of hard plastic? The most obvious answer is the material stiffness, the force needed to bend or deform a material. Most cloth has a very low bending stiffness, imagine placing a t-shirt over a smaller box. Gravity will force the material to bend around the box. Lower stiffness will result in more wrinkles and more deformation and is therefore also key in differentiating the look of jeans and leather from silk or cotton. [i]

Vegetation have similar properties, but is more defined by its ability to spring back to their original shape if forced to bend. Creating a deformation requires a force to be applied, this stores potential energy in the deformed object (say a branch) when the force is removed the object will return to its original shape. The force needed to make a certain displacement can be calculated using the shear modulus.

$\Delta x = 1/S \; F/A \; L_0$

$F = (\Delta x \; A \; S)/ \; L_0$

Where Δx is the displacement from resting position, L is the material's shear modulus constant, A is the branch thickness and L_0 is the branch length. This means that the longer and thinner the branch is the easier it bends. [ii]

A simplification of the tree-scenario would be a flagpole blowing in the wind. What is happening here is that the wind is applying a force to the flagpole. However, because the flagpole is fixed to the ground it doesn't move. It instead has an equal force in the opposite direction at the bottom. This creates a shear force and the pole bends, storing potential energy according to Hooke's Law. When the wind either changes direction slightly or reduces a little in speed the pole will whip back.[iii] A tree is essentially many poles like this connected, affecting each other.

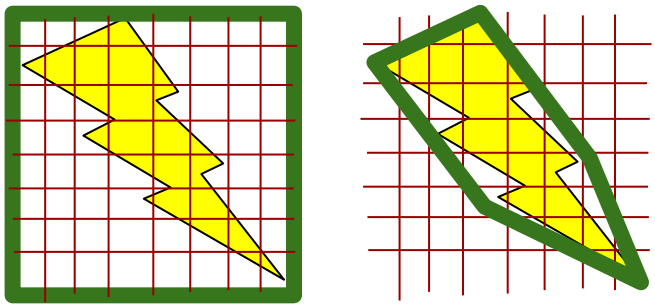
Gravity is very relevant in the hanging cloth scenario. Gravity will force the cloth that isn't fixed to drag down causing wrinkles and tension over the surface. The wind then applies a force to the fabric, causing it to bend away. In this case the bending stiffness is so low that it isn't the force that causes it to go back to its resting position when the wind stops. Instead it is gravity in relation to the tension from the top that complements the wind force. [iv] The tension in the cloth is not evenly distributed and runs in multiple directions. This, together with wind turbulence, causes the cloth to move unevenly in the wind.

Wind turbulence is the unevenness of the wind direction, and it's caused when the wind blows around objects making the wind travel at different speeds. In cities around houses wind turbulence is very high because there is so much disturbance to the wind (i.e. buildings).[v]

2.2 Limitations of mobile platforms

A GPU is created to calculate computer graphics, but can do other kinds of calculations too. The GPU is optimized for SIMD - Single Instruction Multiple Data, that is doing a very specific task on many inputs. This can for example be a vertex program in a shader that needs to run once for every vertex or a fragment program that needs to run once for every fragment.
Due to its structure it's also able to do very fast vector and matrix math, as it's built to work with graphics which is always using colors and transforms.
One thing that really slows down the GPU is complex branching created by using if-statements, because then it's no longer running just one instruction. [vi]

Some modern games use the very specific computation power of the GPU to help relieve the CPU. This is how for example Nvidia PhysX can do really accurate physics simulations in real time. The problem with this is that they only work on certain graphics cards, and certainly not mobile graphics cards!



By adding vertices (green lines) you dramatically decrease the tiles (red) that needs to consider alpha

Mobile CPUs and GPUs are developing very quickly, and getting better by the day, but one of the biggest problems in mobile graphics is the thermal constraints and power consumption. When sending data back and forth the device gets hotter as electricity is converted into heat, and mobile phones are very cramped and don't have room for fans and other cooling that desktop and laptop computers has. As the device gets hotter the clock speed of the GPU will slow down and your game will start running very slowly. The main contributor to this is vertex load and texture look-ups, so on a mobile device you need to keep these to a minimum. [vii]

The main goal of the mobile GPU is to keep power consumption to a minimum, because a mobile phone runs of a battery that needs to last for a long time. The main contributor to power consumption is the video memory. This is why most mobile graphic cards use a tile-based architecture, instead of rendering the image in one go it handles it in tiles usually about 16x16 pixels in size, this way it just has to keep one the tile it's working on in memory. [viii] [ix]

Shaders that use transparency is a big slowdown on mobile graphics cards, because of the tile based architecture and limited memory. The tile based architecture also means that it's often much more efficient to use more vertices to closely follow the alpha border of the texture because this will reduce the amount of tiles that needs to overdraw. [x]

There are two types of alpha shaders, alpha-blended and hard-edge alpha. On most graphics cards hard edge is faster, because it ignores the alpha areas and discards them, so it doesn't have to do all fragment calculations on those fragments.[xi] One interesting exception is these tile based GPUs, most notably the line of PowerVR graphics cards which is used in all Apple mobile devices and many Android devices. [xii] The PowerVR Performance recommendations has the following to say about alpha testing/discard:
"When an alpha tested primitive is submitted, early depth testing - such as PowerVR's Hidden Surface Removal (HSR) - can discard fragments that are occluded by fragments closer to the camera. Unlike opaque primitives that would also perform depth writes at this pipeline stage, alpha tested primitives cannot write data to the depth buffer until the fragment shader has executed and fragment visibility is known. These deferred depth writes can impact performance, as subsequent primitives cannot be processed until the depth buffers are updated with the alpha tested primitive's values." and also notes that *"For optimal performance, consider alpha blending instead of alpha test to avoid costly deferred depth write operations."* [xiii]

You need to keep this in mind when making foliage shaders, which in most cases require transparency, the best option is to make both the discard and blend shader and switch between them depending on the device and cases.

2.3 Common solutions in real time graphics

What are the most common solutions to making animated grass and foliage in games?
One solution is simple joint animation. For example to animate a flag or branch by hand to simulate wind. The downside of this is that it's not very general and has to be done for every object that is to be animated this way. On the other extreme is a completely dynamic solution for everything. In that case everything will be unique, but, this is unrealistic even on the most high end computer today. A third technique that is very common on foliage is animated shaders, just by adding a value to the vertex positions before drawing them on the screen.[xiv] Modern games usually use all three of these techniques in different places.

2.4 Case Study

The first scenario is an object animated to react to wind. Very impressive simulations of this can be seen in some games, like the grass in Flower [xv] or the cloth in Mirror's Edge[xvi] but a mobile device lacks the processing power to do this, and it's usually too intense for most pc and console games too. Instead this is commonly done by using some kind of sinus wave to transform vertices in a shader. The GPU is very effective doing vector math and the vertex program is already looping through all of the vertices to convert them to screen space and send them to the fragment program. Before this we can simply add a value to these positions before sending them down the line. [xvii] [xviii] By supplying properties for the material stiffness wind direction and wind speed a more realistic look can be achieved which correlates more to what actually happens in nature. A second property of how attached parts of an object is could be supplied through vertex color, like in the flagpole example earlier, the bottom of the pole is fully attached while the top moves with the wind.

The second scenario relates to vegetation, for example bushes or grass, reacting to touch as the player walks through them. This is often approximated by using collision detection, and wiggling it a little using rotation values of the object transform. Another method could be using collision and a blend shape for when the player is standing in the bush.

One problem with these methods is that you do not want unnecessary collision bodies, and usually bundle environmental props together into patches of plants instead of single objects, so you can't have a script for each object that shakes the object a little in that case.

Completely simulated foliage as particles is certainly possible, but very expensive. [xix]
CryENGINE uses a simulated joint solution for touch bending of plants, this is costly but a viable method for bigger plants that aren't placed very densely. [xx]
By instead building the bending properties into the shader you wouldn't need any collision detection for each individual plant, instead just look at the distance between vertex and player and scale the bending based on this.

Due to the popularity of the shooter genre in video games, effects relating to shooting has been thoroughly researched over the years. However, not many games has cloth that reacts to bullets due to the complexity required to do this. Some titles like Borderlands 2 and Mirror's Edge uses PhysX for real time cloth simulations that supports tearing by bullets, but this only works with Nvidia graphics cards.

The effect can be divided into two main components. First the cloth reacts to the bullet by being forced to move back with the bullet creating a stretching and movement in the cloth. The second part is that the bullet leaves a little hole and the cloth moves back into resting position as the force is no longer applied. The second part is the actual tearing where after multiple gunshot the cloth actually breaks into pieces which falls to the ground or hangs to the side.

If the second part is disregarded we can easily emulate the solution by just supplying a shader with the vector of the shot and a couple of points along the vector. The shader can displace vertices close to the bullet position in the direction the bullet is travelling and then fall back as it passes. For a more complete illusion a particle system could spawn a hit effect and the model could be swapped for a damaged version.

3 Method

3.1 Implementation

I implemented a solution for each of the cases in Unity, writing the shader code in Cg. The shader files are included as an attachment. When writing about the cases I leave out all unity specific things, and the information should easily apply to any game engine and shading language.

Sheets blowing in the wind, note patterns of the wrinkles and wave-like movement
Image by: Jonathan Abarquez

3.1.1 Reacting to wind

This is a relatively simple solution, and it can be contained entirely in shader code. The vertex program takes an input-wind in the form of a vector 4 where the first 3 inputs are the direction vector (with displacement scale as magnitude) and the fourth is the wind speed. There is another input called stiffness which roughly maps to how a fabric stiffness and weight affects the way it moves. Higher stiffness means it moves less overall and also has less small wrinkles in the movement.

The amount of displacement the wind should do on the object is then calculated by multiplying the wind direction by 1/stiffness. Finally it is multiplied by vertex color representing how loose that vertex is, for example the area were a bush is anchored to the ground shouldn't move as much as the top.

```
disp = (_WindDir.xyz*(1/_stiffness)*v.color.r;
```

Now we add that displacement to the vertex position before sending it to the fragment program. We multiply it by the sine of time offset in y by 0.5 and divided by 2 so it moves between 0 and 1. What this means is that the displacement changes between no displacement and full displacement.

```
pos += disp * (sin(_Time*_WindDir.w + waveOffset).w/0.5+0.5);
```

The frequency of the waves is scaled by the w component of the wind direction, the wind speed. A wave offset is added for every vertex. This wave offset ensures that the entire object doesn't move as one, there is actually a wave pattern going on, like in the image to the right. The offset is calculated like this:

```
waveOffset = (pos.x + pos.y + pos.z)*(2/_stiffness);
```

This means that an object with higher stiffness, like leather, would move with less variation (bending, wrinkles) than one with lower stiffness.

Finally a pass of more intense, more unpredictable movement is added. It uses vertex coordinates to offset the time and scale so that it breaks up the movement a lot. It is multiplied by the green vertex color channel. This Looks best along very loose edges of cloth or around the edges of leaves. Therefore it's masked by the green component of the vertex color so the affected area can be painted individually from the rest of the mesh.

```
noise =  sin(_Time*_WindDir.w*2.372*frac(pos)).z*frac(length(pos))*v.color.r*(1/_stiffness)*v.color.g;
pos += noise;
```

3.1.2 Reacting to touch

The goal of this is creating grass or bushes that moves and bends slightly as you run past. The displacement effect is done in the shader much like in the previous case, and can be combined with wind to make a more interesting shader. The reaction is done by supplying the material with the velocity and position of the player in an external script attached to the player.

The vertex program first calculates how much the vertex should be affected by the player based on how far away it is. First it gets the distance between the positions then saturates the max influence distance. The saturate function returns 0 if the input is less than 0 and 1 if it's more than one.

```
dist = distance(o.pos.xz, _CharacterPosition.xz);
mixFactor = saturate(maxDist-dist);
```

The touch function the pushes the vertices in the direction of the character movement and slightly down. This transformation is multiplied by the length of the movement vector. This means that as the character slows down the vertices will spring back to their original shape.

```
touch = (_CharacterMovement+ float3(0.0,-1,0.0)*length(_CharacterMovement));
touch = touch * mixFactor *v.color.g *(0.2/_stiffness);
```

The whole thing is then multiplied by the mixFactor, the green vertex color which masks the influenced area and a constant divided by stiffness.

This whole thing can easily be built upon the wind shader for a more comprehensive solution.

```
pos += float4((wind + touch + noise)*v.color.r,0);
```

3.1.3 Reacting to bullets

This is essentially the same as in the case above.
First the vertex mix factor is calculated to see if the vertex should be affected by the bullet.

```
dist = distance(pos.xz, _ShotPos.xz);
mixFactor = saturate(maxDist-dist);
```

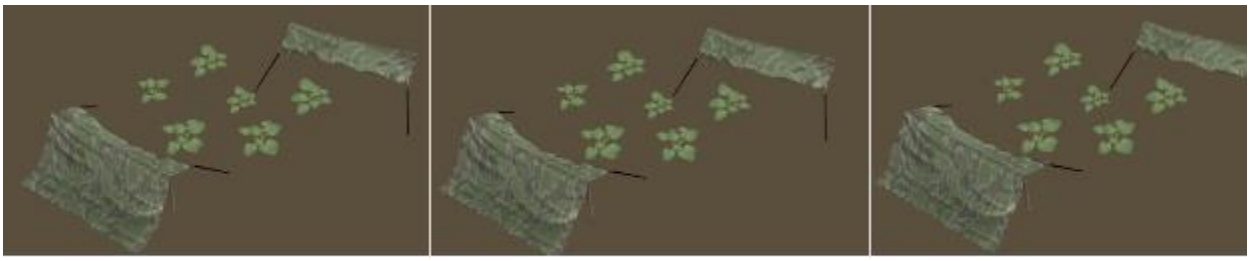
Next we have to figure out how much displacement should take place. This is based on displacement value given from the weapon, _ShotVector.w, and the material stiffness. The position is then offset along the shot vector by the shotDistance and masked by the mixfactor and the red vertex color.

```
shotDisp = _ShotVector.w * (1/_stiffness);
pos += float4(_ShotVector.xyz,0.0)*shotDisp*mixFactor*v.color.r;
```

This can be combined with wind effects and touch effects for a more complete look.

4 Result

The key property to defining these materials turns out to be how they react to external forces. Different materials are defined largely by stiffness. In my shaders I decided to use this property in most animations for many different functions from how far vertices are displaced to how much internal variation there is.
Here is my final wind shader. It is quite hard to make out the animation these still frames, but it can be seen on the left side of the leftmost tent.



Example of the wind shader

Here is an image of the touch reacting, a foliage shader reacting to the player walking through it by bending down and moving a little in the player walking direction.

Vegetation Shader, reacts to player walking through it

And here is an image of the shot reactive shader

Material reacting to shot (white line)

The final shaders are written in Cg (included as an attachment), ready to be used in unity but could easily be translated into other shading languages and game engines. A video recording can be seen on http://www.linuslindberg.com/p/thesis.html.

After looking into the performance constraints, I quickly disregarded any actual vertex simulations, which are popular in PC and console games as they are too performance heavy for mobile devices. The choices left were joint animation or a shader solution.
Instead I decided to go with a shader solution. The result was a solid wind shader (though not very novel), an interesting touch reaction shader and a bullet reaction system with potential.

Here is a comparison with the wind shader (C) and two other common solutions, simple joint animation(A) and a real-time simulation (B). The test is done on a scene with 160 objects that require the wind effect, and tested on different devices.

	iPhone 6	iPhone 4s	MacBook Pro
(A) Animated Joints - 3 joints	57.1 fps	27.3 fps	54.1 fps
(B) Unity Cloth	25.2 fps	0.4 fps	29.1 fps
(C) Wind Shader	55.1 fps	57.0 fps	54.2 fps

Here we can see that the animated joints with a basic unlit shader are slightly faster on the more powerful devices, because here the cpu can keep up with the graphics and it's the rendering step that causes the slowdown. This means the more complex shader, the wind shader, is going to take the longest to render. On the older phone however it's the cpu that limits the framerate, which will make the wind shader almost twice as fast because it's very light on the cpu. The Unity cloth is very slow on the iPhone 4s with a frame every other second. On the iPhone 6 the performance is much better, at about 25 frames per second, but still to slow for just a wind effect.

Visually the wind shader has more variation than the animated joints because it ignores the object rotation, and changes the offset based on position. The cloth simulation has potential for even greater variation and interesting visuals, because it's fully simulated.

The difference between my wind shader and my reactive materials in performance is almost none, It's just adding a few more instructions in the vertex shader.

5 Analysis and Discussion

Looking at the final result, the first two shaders are solid, and almost production ready, but the shooting example doesn't really look good enough. To me the shooting solution was an interesting experiment. There is some potential in it, but there is a lot of additional work required to use in a real game. One could for example supplement the effect with particle burst to make it more rich and visually appealing.

One big problem with my solutions for problem b and c is that they only take one character and one bullet into account, because they only have one vector input. This means that in a multiplayer game you can only see when you yourself run through grass or shoot at things. You could add one input for every potential player, but for every added player there would be multiple lines of shader code added, which all needs to run once for every vertex. This might cause slowdowns, which need to be tested further to see if the solution is still viable in this case.

For a more natural look a script could be written to dynamically change the wind direction and intensity over time and supply that value to the material instead of just a static vector. In the real world the wind never blows in the same direction for very long.

The shaders for wind could easily be improved for little extra cost by using all four channels of the vertex color for other properties instead of just two, like using the blue for time offset and alpha for stiffness variation. This would allow for greater control for the artist, and more realistic looking object in many cases.

A variation on the shot-reactive material with even better performance and slightly nicer looking movement could perhaps be created by instead of sending multiple positions along the shooting vector just sending the player position and the shooting vector. The shader would then create a new vector from the player position and the vertex position. The dot product of this new vector and the shooting vector would be used to calculate which vertices should be affected by the shot.

In conclusion I have to say that I like the idea of containing these effects in a shader. The wind effect is something I've seen done many times but providing other input for additional forces, say the player velocity, does add more of an interactive feel to it at a low cost. As I've stated earlier, the shot reactive material still misses some parts to make it more appealing, and is more of an early idea than a full implementation. Nevertheless I feel exploring this further is something that could add a lot of immersion to mobile games especially.

References

[i] "Influence of the fabric properties on fabric stiffness for the industrial fabrics" Mehmet Emin Yüksekkaya, Thomas Howard, Sabit Adanur

[ii] "College Physics" "5.3 Elasticity: Stress and Strain" Dr. Paul Peter Urone Dr. Roger Hinrichs

[iii] "College Physics" "16.1 Hooke's Law: Stress and Strain Revisited" Dr. Paul Peter Urone Dr. Roger Hinrichs

[iv] http://www.physicsclassroom.com/class/waves/Lesson-0/Pendulum-Motion (April 21 2016)

[v] http://www.greenrhinoenergy.com/renewable/wind/wind_characteristics.php (April 28 2016)

[vi] "From Shader Code to a Teraflop: How Shader Cores Work" Kayvon Fatahalian, Stanford University, SIGGRAPH 2008

[vii] The Revolution in Mobile Game Graphics (Presented by ARM) - GDC 2014 - Marius Bjorge, Sam Martin

[viii] The Revolution in Mobile Game Graphics (Presented by ARM) - GDC 2014 - Marius Bjorge, Sam Martin

[ix] Broken Age's Approach to Scalability - GDC Europe 2013

[x] Broken Age's Approach to Scalability - GDC Europe 2013

[xi] https://en.wikibooks.org/wiki/GLSL_Programming/Unity/Transparent_Textures (April 22, 2016)

[xii] https://en.wikipedia.org/wiki/Apple_mobile_application_processors (April 22, 2016)

[xiii] http://cdn.imgtec.com/sdk-documentation/PowerVR+Performance+Recommendations.The+Golden+Rules.pdf section 2.8

[xiv] http://http.developer.nvidia.com/GPUGems3/gpugems3_ch16.html

[xv] http://www.thatgamecompany.com/forum/viewtopic.php?t=2296

[xvi] http://physxinfo.com/wiki/Mirror's_Edge (May 22 2016)

[xvii] "Animated Grass with Pixel and Vertex Shaders" John Isidoro and Drew Card

[xviii] http://www.gamasutra.com/view/feature/132419/sponsored_feature_rendering_grass_.php (March 12 2016)

[xix] "Real-Time Simulation of Interactive Grass Mesh" By Godwill Fameyeh

[xx] http://docs.cryengine.com/display/SDKDOC2/Touch+Bending (March 12 2016)