Grass Effect on Teapot

Shanshan Cai
USC Viterbi School of Engineering
Los Angeles, CA
cais@usc.edu

Lineng Cao
USC Viterbi School of Engineering
Los Angeles, CA
linengca@usc.edu

Sifan Geng
USC Viterbi School of Engineering
Los Angeles, CA
sifangen@usc.edu

***Abstract* - This paper reports a project inspired by a Unity grass texture implement. The goal of the project is to mimick the blades of grass effect and bring into GzRender homework application for the outer surface of the teapot. This paper also tries several ways to make the best output effect. The grass is naturally bent under wind effects of normal mapping and rendered with shadows effect by using depth mapping as well as two-pass z-buffer.**

Keywords - grass effect, rendering, rasterization, shade, texture mapping, normal mapping, depth mapping

I. INTRODUCTION

Hair or fur rendering is always considered as a major topic in rendering and rasterization. There are already numbers of existed high-performance libraries in DirectX 11 and OpenGL can handle the rendering process easily while effectively. With the inspiration of one open-source fur rendering library based on Unity and grass geometry shader algorithm from Roystan, the grass effect can be created in two different ways. A certain width, less density, and a more bent algorithm is needed; thus, this report chooses the second algorithm method combining as an enhanced GzRender application [1][6]. This report uses open-source fur shader in Unity while mimicking real grass effect by manually programming grass generator algorithms, including grass bending, normal mapping, two-pass z-buffer, and depth mapping under GzRender application, a previously modified homework project including basic file reading, file writing, rasterization, geometric translation, and anti-aliasing.

A. Open-source Fur Rendering Shader

The open-source fur shading in Unity this report takes is a Unity version of the WebGL implement method, based on the author's introduction [6]. Behind the code, it brings an extra soft feeling of the object by adding multiple layers and make hairy. The article pointed out that WebGL using bump mapping to increase hair or fur realistic effect, but it was not merged in this library. Two maps were defined in the C# file in this library, one as hair or fur texture, and one as noised cross-sectional texture, as shown in Figure 1. The hair or fur texture was used for the ground image of the object as normal texture, while noised cross-sectional textures became one of the multiple layers for achieving its fur effects. Multiple passes were also needed in Unity. The author used hard-coded 60 times in code to different FUR_OFFSETs [6]. Considering the algorithm while its final effect, this paper does not use the idea behind this method of rendering.

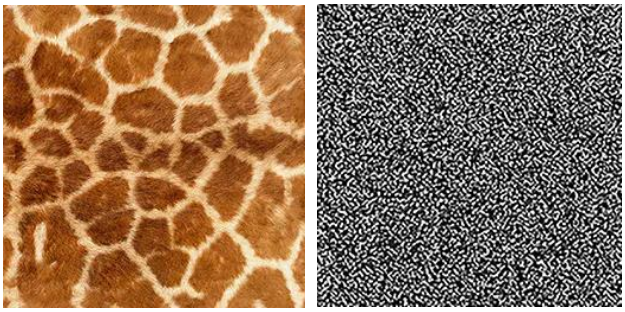

Fig. 1. **Left**: Fur texture used in Fur Rendering Shader [6], **Right**: noised corss-sectional textures used in Fur Rendering Shader [6].

B. Grass Geometry Shader algorithm

Roystan's article of grass geometry shader provided one Unity instance of demo project but also showed the algorithm of geometry modification of grass on original object [1]. The idea of the grass geometry shader is recursively creating grass on the existing "ground" object and combining as a whole object for shadowing and other wind movements. The article introduced the bend algorithm with random seed involved. The rest of the report focuses more on the algorithm level of Grass Geometry Shader and codes in native C++ way in our GzRender application.

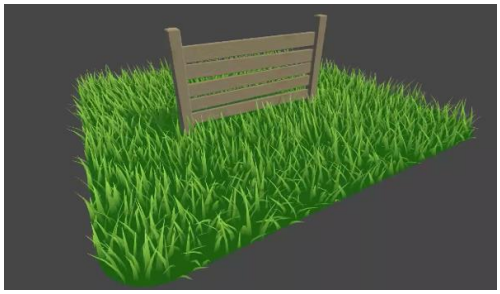

Fig. 2. Sample effect of Grass Geometry Shader algorithm [1]

II. METHODOLOGY

This report analyses two algorithms by implementing both in real code condition listed as below.

A. Implement Grass Effect in Unity

The Open-source Fur Rendering Shader is used in the Unity version of the grass effect. Mainly four parts are taken into transferring fur into grass render [6].

1) Import library and teapot object

The current fur-render library is imported as an existing fur rendered sample from the library GitHub page, which shows one

way of turning an already imported object into the fur effect. One fur material is attached to its object with custom shader and shader helper code files. Teapot data from homework is been converted into an obj format file and imported as an external model

2) Create texture and customized materials with fur shaders

The fur features are realized by specific textures, an image of fur in the sample. To display the vivid grass effect, one grass photography is imported as a texture. The project replaces the grass texture image with one map in the code and keeps the noised cross-sectional textures as original, then drags new generated material as prefab in the unity project for modularized purpose.

3) Attach customized materials with shader on teapot

The reporter adjusts the scale factor of the teapot object for matching the global coordinates and brings the best grass effect. It is not necessary to store as prefab, but in that way can keep the structure cleaner. The next step is dragging customized material into the newly created teapot object. Underneath the codes, the shader and shade helper are attached to the teapot object since Unity provides handle dragging feature to make binding action easier and more straightforward.

4) Grass effect

The code allow user to tweak properties like roughness, offset tiling and gravity in the shader class though Unity side panel inside each material. With minor modifying the value of properties, the grass effect of the teapot is shown in Unity, e.g. grass effect in Figure 3.

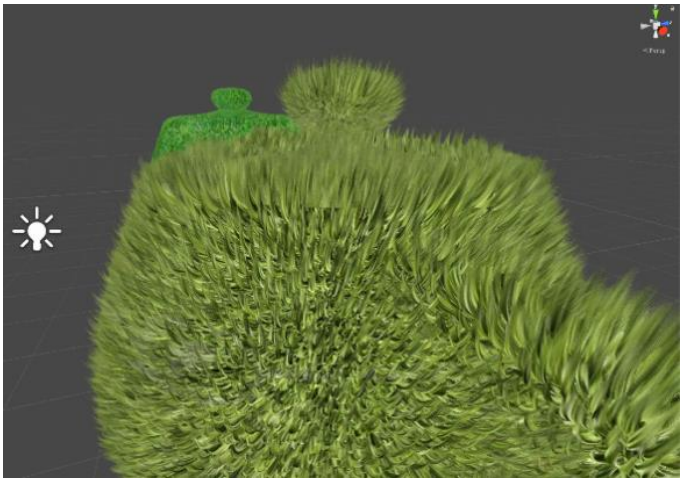

Fig. 3. Grass effect in Unity by using Open-source Fur Rendering Shader

B. Implement Grass Effect in GzRender Application

The Grass Geometry Shader algorithm is used in GzRender Application. Since the original sample of the article was coded in Unity with C#, this implement is pure C++ coded on previous work of GzRender Homework without any extra library help. The extension of Grass Geometry Shader algorithm is coded into C++ with GzAddGrassWithModelSpace(), GzCalShadowDepth(), and GzPutTriangle().

1) Modifying teapot with attached grass information.

The main idea of modifying teapot with grass information is attaching extra information during each line of triangle surface while reading from *Application.cpp*. In this enhanced GzRender with grass effect, the original input file, "ppot.asc" is not changed or altered. Thus, all the information about the grass is not able to store into a file, they are ad-hoc and are kept in memory during software running for further modification or recursion.

The main part of generating one single grass can divide into two steps. Finding one mid-point of a triangle surface and creating extra triangles based on that mid-point. The new generated grass holds its specific tangent space and the lookup direction of tangent space is the normal direction of the mid-point. Some demonstrations are shown in Figure 4.

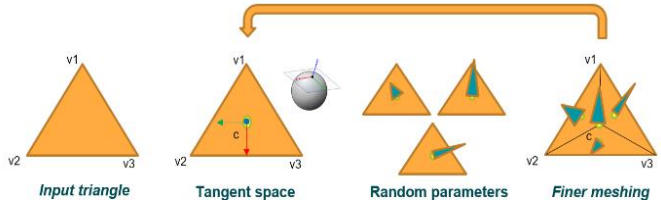

Fig. 4. Demonstrations of tangent space for grass attached object, and recursion involved

a) *Find middle points as a base for triangles of grass (Centroid or Incentre of triangle):*

The selection of mid-point to a triangle can be various, it can be incentre, centroid, circumcenter, orthocenter, nine-point-center and so on. This report mainly focuses on choosing the incentre and centroid of a given triangle surface and more discussion is mentioned in the recursion part.

After the mid-point is chosen, one triangle regard as the grass is added, and the current mid-point recursively joins with any two of vertexes in the original triangle make mid-point' again, and so on. Thus, with one loop of full recursion, each triangle brings four extra grass information into the code. those positions depend on the input triangle's vertexes. The mid-point needs to always locate as inside of the triangle no matter for obtuse or acute triangle and keep its x, y, z within given triangle vertexes. Thus, if attach a grass is based on the middle point, it will always within the bounder.

b) *Create extra triangle information on each existing triangle before GzPutTriangle:*

The top point represents the height of the grass on the teapot. Before *GzPutTriangle()*, extra grass-triangle information should be added on each initial triangle, the information includes position (x-axis, y-axis), height and width.

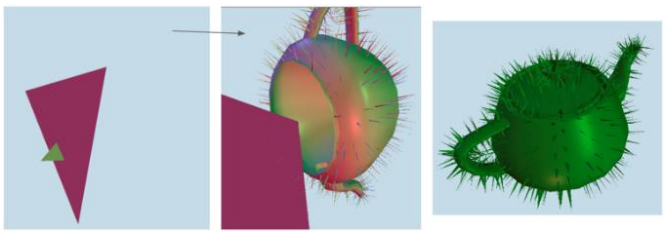

Fig. 5. Grass created on single triangle and extend to teapot in model space. **Left**: Single striaght grass is implemented from a mid point of a triangle

surface. **Middle & Right**: grass are generated with each given triangle from ***'ppot.asc'***.

2) Create grass with bend effects & recursion of grass generator

a) Bend effects:

Bend grass effect is closer to the natural appearance of the real grass. If the blades of grass all stand up perfectly straight, they look very identical and lack of realistic. This may be desirable for well-tended grass, like on a putting green, but does not accurately represent grass in the wild. Thus, the algorithm of bent grass is designed. The grass triangle is divided into 5 pieces with different *u*, *v* and coordinates in tangent space. Also, random functions are used for randomly create the degree of bend pieces. The exponential blade curves are added to the grass model associating with the value of *z*. All are calculated by codes.

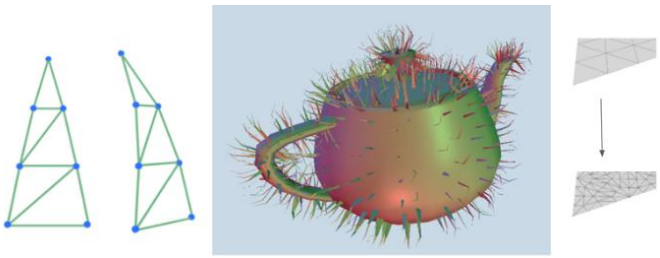

Fig. 6. Bended algorithm of recursive creating bended grass. **Left:** Five triangles form each single bended glass with exponential blade curve [1]. **Middle:** images shows output of bened grass effect. **Right**: image shows idea of recursive way to exponential increase grass number [1]

b) Recursion in striaght:

The first grass is attached in the middle point of the input triangle, then the lines between the middle point and each of three input triangle vertices separating the input triangle into 3. Recursively dividing and appending in each of those middle points. After 4 to 7 loops, recursion brings great grass effect with straight grass. The selection between centroid and incentre influences the final result. Since the Incentre center point is more likely to allocating in one certain area, while the Centroid center point is uniformly distributed. This report uses the Centroid center point for mid-point with the rest of the discussion.

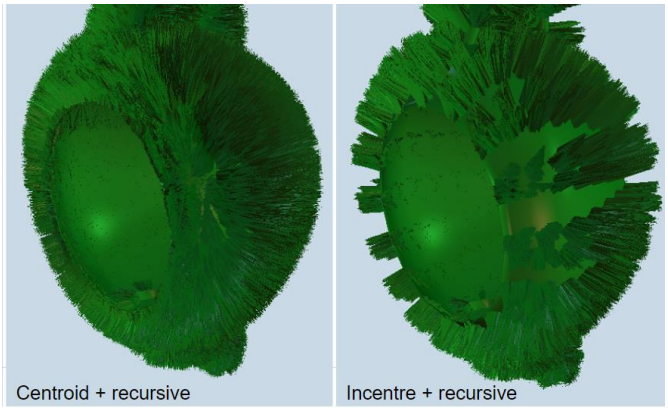

Fig. 7. Sample outputs of stright grass effect. **Left**: Camera 1 with centroid of triangle as midpoints with recursion (4 loops) invloved grass generator. **Right**: Camera 1 with incentre of triangle as midpoints with recursion (4 loops) invloved grass generator

c) Recursion in bend:

Recursion in bend grass is mostly like recursion in straight grass. However, instead of storing one single triangle, 5 triangles in one single grass are needed to store. The result is satisfying compared with the straight grass effect. It mimics the real grass in different camera positions. Also, for preventing showing inconsistent of the density of grass, apart from loop counts, the distance between mid-point and any one original vertex is checked during recursion and it will break the loop once a certain density (distance <= 0.1f) is reached.

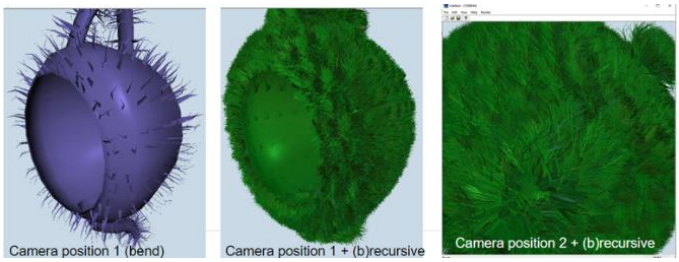

Fig. 8. Sample outputs of bended grass effect. **Left**: Camera 1 with once time grass generator. **Middle**: Camera 1 with recursion (4 loop) invloved grass generator. **Right**: Camera 2 with recursion (4 loop) invloved grass generator

3) Set u, v in Grass

a) Grass shader - u, v, with texture mapping:

Instead of the procedure color of the teapot with grass as a green object, the report chooses UV texture mapping first. By replacing texture images with photography of grass, the grass showing color-shifting based on the texture. It does show some effort similar with output from Unity with open-source fur shade library.

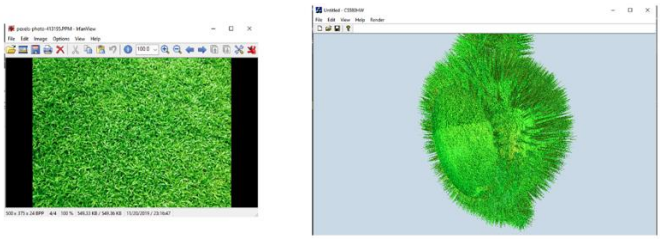

Fig. 9. **Left:** Simple texture by using the photograhy of grass [5]. **Right**: render *tex_fun()* with basic UV mapping. **Right**: render *tex_fun()* with basic UV mapping

b) Grass shader - u, v, with procedural texture:

Using a procedural texture is another way of mimicking the grass color in realistic. The idea of bringing great procedural texture to the grass is separating the ground color and grass color. The grass can use a color gradient to differentiate the different colors between the root of grass and top of grass. This project uses an extra step of checking if the triangle is in the grass by setting a unique value in *w*. If the triangle does in the grass, a color linear gradient starts to change based on *u*, else, all ground triangle will have a solid dark color texture.

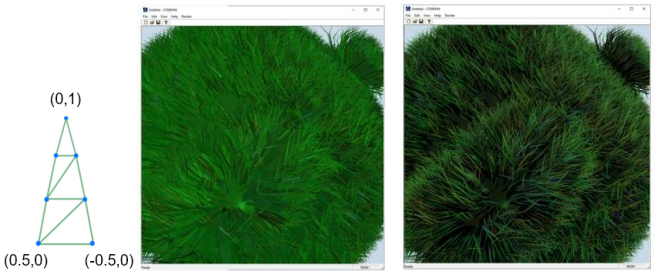

Fig. 10. **Left:** Implement of (*u*, *v)* in grass tangent space, creating color gradient, render p*tex_fun()* with procedural texture. **Middle**: Teapot without color gradient. **Right**: Teapot with color gradient

4) Generate wind distortion pattern using Normal mapping

Wind effect is also been designed by sampling a distortion texture. This texture matching with a normal map, with two channels (red and green). The report uses two channels as the x and y directions of the wind. The grass bends based on the pre-designed normal direction.

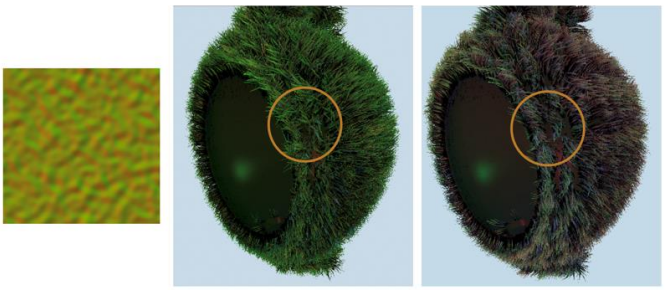

Fig. 11. **Left**: Implement of 2-channel "normal mapping", **Middle**: Teapot without Normal Mapping, **Right**: Teapot with Normal Mapping

5) Implement Depth Map and Two-Pass Z-Buffer Shadows effect

Shadow is normally used to reflect the perception of depth and distance of an object. Shadow effect is also designed for mimicking the real grass. In this project, Two-Pass Z-Buffer algorithms are used to create shadow map. The Image-space shadow calculation is coded for generating depth maps of the teapot. The idea of the algorithm is rendering z-buffer depth image from light source. Since the light in the original GzRender is parallel light, the project temporally created light position and the field of view (FOV) as same as camera position1, while keeping light original Lookup direction. After a screen image from light source is generated as shadow map, computing a invert view-camera transformation and compared with depth map with z depth in each pixel can identify whether this point is in shadow or not.

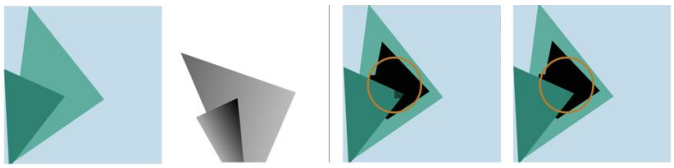

Fig. 12. Triangles without shadow, Depth Map, Hard shadows (shadow acne) & Soft shadows (additional bias)

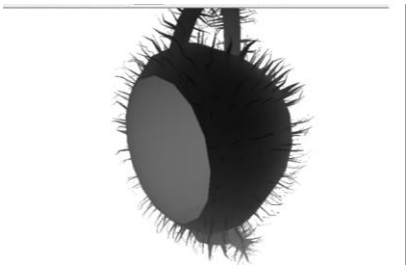

Fig. 13. One depth map of teapot with bend grass effect modified object of teapot with bend grass effect

III. DISCUSSIONS

There are several existing libraries for more enhanced hair and fur rendering. The purpose of this project is using the basic concept of algorithms to build the grass effect from scratch. The GzRender Application is already contributed by authors with rasterization and geometry transformation, thus it is a perfect initial project for this report. The newly contributed method, *GzAddGrassWithModelSpace()*, *GzCalShadowDepth()* and *GzPutTriangle()*, inside rend.cpp are built for delivering the grass effect in high performance. Figure 11 shows a convincing result of grass effect with Phong shading, while still holding the function of scaling, rotation, and translation.

IV. FUTURE WORK

A. Shadow

The final shadow effects of the teapot are still not in perfect output, Figure 14 indicates two failure outcomes, based on two depth maps with different light directions. Since the report does the z-interpolation and two-pass z-buffer for getting an inversed z depth value, which might causes the issues inaccuracy. In both outputs, the inner interface brings the inaccurate shadow results.

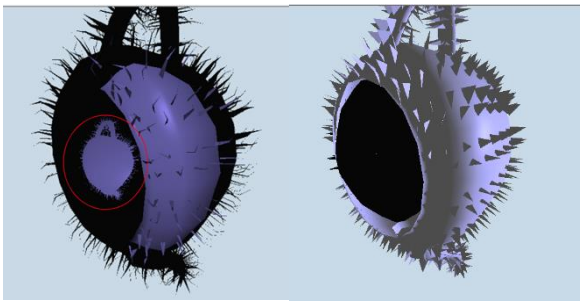

Fig. 14. Failure shadow ouput 1 and 2 with inner interface of the teapot

B. Animation

This project does bring a gif format animation image of grass growing effects on the surface of the teapot. A timer slider is added in the GzRender application to control the length of grass-based on time. The gif file is generated by several images using extra tools. The next step is enhancing fully capability for exporting animation image(gif) files or video files in GzRender. An animation of either wind effects or grow effects can be selected based on user preference.

C. Anti-aliasing

Anti-aliasing is not implemented in current status. The report aims to focus on grass and shadow first. Anti-aliasing is the future work this report will work on as well as bringing much-optimized grass effects.

V. REFERENCES

[1] "Unity Grass Geometry Shader Tutorial at Roystan", *Roystan.net*, 2019. [Online]. Available: https://roystan.net/articles/grass-shader.html. [Accessed: 31- Oct- 2019].

[2] Cohen et al., “Appearance-Preserving Simplification”, SIGGRAPH 1998.

[3] L. Williams, "Casting curved shadows on curved surfaces", *ACM SIGGRAPH Computer Graphics*, vol. 12, no. 3, pp. 270-274, 1978. Available: 10.1145/965139.807402.

[4] edom18, “edom18/Fur-shader-sample,” *GitHub*, 29-Sep-2019. [Online]. Available: https://github.com/edom18/Fur-shader-sample. [Accessed: 01-Dec-2019].

[5] Gravitylicious.com, *Green Grass*, May 2019

[6] Hiruma@edo, Kazuya. “[Unity] ファーシェーダを移植してみた.” *Qiita*, 20 Sept. 2014, https://qiita.com/edo_m18/items/75db04f117355adcadbb