Procedural Modeling and Shading of a Golf Ball in RenderMan

Anu Kriti Wadhwa
s5647918@bournemouth.ac.uk
Bournemouth University
Bournemouth, United Kingdom

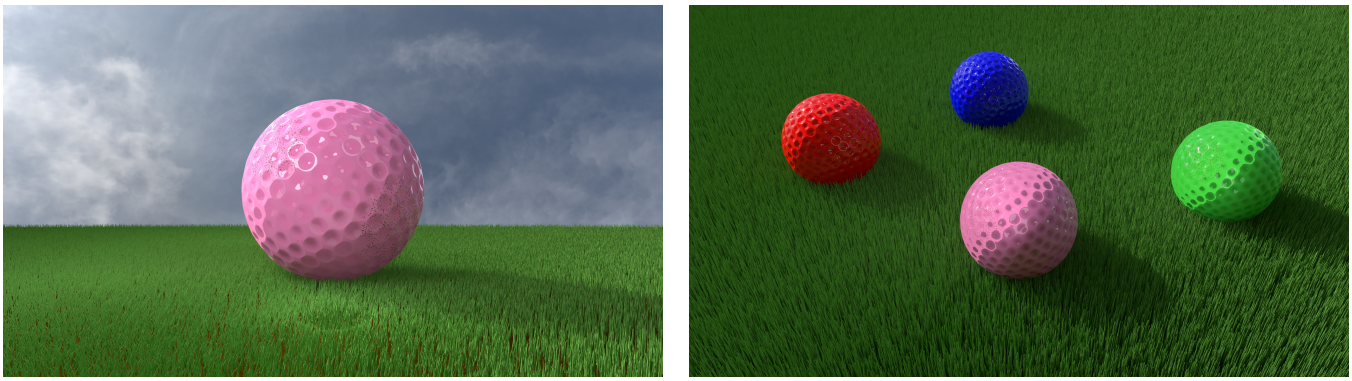

Figure 1: Final render images. (a) A close-up front view of a single pink golf ball on a grass field. (b) An angled top-down view showing four golf balls of various colours on a grass field.

Abstract

This project explores the use of procedural techniques in 3D modeling, shading, and scene composition using Pixar's RenderMan, Python, and OSL shaders [4, 9, 12]. The goal was to generate high-quality still renders of a golf ball scene without relying on digital content creation tools. To model the golf ball's dimples, a Fibonacci sphere algorithm was employed, offering a plausible approximation of the dimple pattern. Shading was enhanced with displacement and shadow tinting. Procedural dirt and wear effects added realism. Scene composition included procedural grass generation using a patch-based approach with distance-based density falloff, as well as the instancing of multiple golf balls with varied colours and rotations. The final renders — a single golf ball close-up and a multi-ball scene — demonstrate how procedural workflows can achieve visually compelling and technically controlled results.

CCS Concepts

• **Computing methodologies** → **Rendering**; • **Applied computing** → **Media arts**.

Keywords

RenderMan, Shading, Look Development, Physically Based Rendering, Material Networks, Production Rendering

1 Introduction

The use of procedural methods in computer graphics enables scalable, reproducible, and highly controllable workflows, particularly when combined with modern rendering systems like Pixar's RenderMan. This report presents a procedural approach to modeling, shading, and scene composition for a golf ball rendered entirely through code. By utilizing RenderMan's RenderMan Interface Bytestream (RIB) interface, Open Shading Language (OSL) shaders, and Python scripts, the project demonstrates the integration of mathematical modeling and procedural texturing in a dynamic scene.

The report begins by describing the challenges of replicating the golf ball's dimple pattern and adoption of a Fibonacci sphere algorithm for approximation. It then outlines shading strategies to enhance realism, including shadow tinting and procedural dirt. Subsequent sections discuss procedural grass generation with performance optimizations and instancing to arrange multiple golf balls with natural variation. The report concludes by evaluating the visual results and highlighting the benefits of procedural workflows for the generation of complex scenes.

2 Modelling the Golf Ball

2.1 Goldberg Pattern

The characteristic surface pattern of a traditional golf ball follows a **Goldberg polyhedral tiling** [3], specifically a **5,1** Goldberg polyhedron (Figure 2). To identify the pattern, a physical golf ball was inspected and marked using tape to trace the number of vertices, edges, and faces that confirm the 5,1 configuration (Figure 3).

While the Goldberg polyhedron provides an elegant geometric framework for distributing dimples evenly across a sphere, it is difficult to implement procedurally. Goldberg structures are not generated through uniform grid subdivision. Instead, they are derived from complex tiling rules over an icosahedral base, where clusters of triangular faces must be grouped into larger polygonal patterns (hexagons and pentagons).

MSc CAVE Rendering Project, Bournemouth University, UK

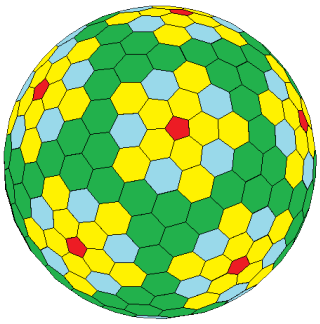

Figure 2: Goldberg polyhedron 5,1. [10].
items

Figure 3: Reference Golf balls. The pink golf ball (left) marked with a red square showing a distinct 'smaller' dimple that occurs 12 times around the ball. The yellow golf ball (right) has been taped to study the pattern of the dimples, with the tape meeting at the small dimple locations.

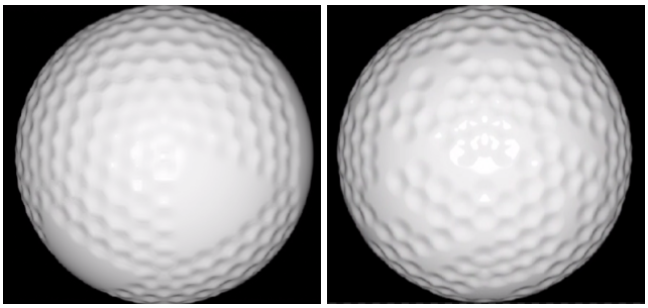

Figure 4: Some failed attempts at procedurally mimicking the Goldberg Dimple Pattern showing gaps and overlapping dimples on the edges.

Attempts to procedurally generate the Goldberg patterns involved using libraries such as `trimesh` and `numpy` to construct base meshes and analyze triangle groupings via centroid clustering and barycentric distance checks. However, this approach yielded incomplete tiling, over-clustering, and visual artifacts such as dimple overlaps or gaps (Figure 4).

The difficulty lay in identifying and organizing triangle patches into coherent polygonal regions that conformed to Goldberg tiling rules. Consequently, a more tractable alternative was sought.

2.2 The Fibonacci Sphere

To overcome the limitations of Goldberg tiling, the project adopted the **Fibonacci** sphere technique, a mathematically elegant method for distributing points evenly over the surface of a sphere [2]. Unlike the Goldberg polyhedron, which relies on a strict polygonal face structure, the Fibonacci approach provides quasi-uniform point spacing that closely approximates even dimple distribution.

The Fibonacci sphere algorithm places N points on the surface of a unit sphere by leveraging the golden angle, which is defined as:

$$\theta = \pi(3 - \sqrt{5}) \approx 2.39996 \text{ radians}$$

Each point $i \in \{0, 1, ..., N-1\}$ is placed on the unit sphere using:

$$y_i = 1 - \frac{2i}{N-1}$$
$$r_i = \sqrt{1 - z_i^2}$$
$$\phi_i = i \cdot \theta$$

The Cartesian coordinates of each point are then given by:

$$x_i = r_i \cdot \cos(\phi_i)$$
$$z_i = r_i \cdot \sin(\phi_i)$$
$$y_i = y_i$$

This results in a spiral distribution of points that avoids clustering at the poles — an issue common to latitude/longitude-based sampling — and provides visual uniformity, making it ideal for the procedural shading and displacement of golf ball dimples.

2.3 Implementation

A Python script was written to procedurally generate dimple center positions using the Fibonacci sphere algorithm (Algorithm 1). These coordinates were exported to an `.oslinclude` file.

Algorithm 1: Generate Dimple Center Coordinates

Input: Number of dimples N
Output: `dimple_centers.oslinclude`
for $i \leftarrow 0$ **to** $N-1$ **do**
  $y \leftarrow 1 - \frac{2i}{N-1}$;
  $r \leftarrow \sqrt{1-y^2}$;
  $\theta \leftarrow i \cdot \phi$; `// golden angle`
  $x \leftarrow r \cdot \cos(\theta)$;
  $z \leftarrow r \cdot \sin(\theta)$;
  append (x, y, z) to list of dimple centers;
Write list of dimple centers to `.oslinclude` file as a function

An OSL shader then reads the dimple centers coordinates and displaces the golf ball surface based on proximity to them (Algorithm 2). For each shading point, the distance to all dimples is checked, and the point that has the minimum distance within range,

a displacement is applied to form the dimple. The minimum distance logic ensures dimples do not overlap.

Algorithm 2: Compute Dimple Displacement

Input: Shading point position `shading_point`; array `dimple_centers`; `dimple_radius`, `dimple_depth`
Output: Displacement amount `displacement`

```
pos ← normalize(shading_point);
min_angle ← dimple_radius;
best_falloff ← 0;
for each center in dimple_centers do
    angle ← arccos(pos · normalize(center));
    if angle < min_angle then
        min_angle ← angle;
        best_falloff
          ← 1 − smoothstep(0, dimple_radius, angle);
displacement ← -(best_falloff × dimple_depth);
```

2.4 Qualitative Differences — Goldberg vs. Fibonacci

Unlike the Goldberg pattern, the Fibonacci sphere does not form true polygonal tiling but creates a visually even, aperiodic dimple distribution. It allows easy control over dimple count, scale, and depth, enabling flexible artistic tuning. This approximation effectively captures the golf ball's texture without the complexity of replicating its exact geometry.

3 Shading and Lighting

3.1 BXDF properties

The primary surface shader for the golf ball was implemented using the `PxrSurface Bxdf` [7]. The material aimed to replicate the look of glossy plastic as observed in the photographic references of real golf balls. After setting the right base colour of the ball, subtle `clearcoat` highlights were added to capture the sheen typically seen under strong lighting.

Figure 5: A photograph of a single red golf ball under the Sun

Figure 6: A photograph of 4 golf balls under the sun

3.2 Shading

In Figure 5, the dimples look flat where the light hits them directly. In Figure 6, however, we see all the dimples distinctly on the golf balls. To see those distinct dimples in the second render image (Figure 1b) without increasing the depth of the dimples, shading-based tricks were introduced inside the OSL shader (Algorithm 3). A `shadowTint` colour output darkens the base colour inside dimples using a smooth falloff from each dimple center. This gives the illusion of depth via lighting rather than displacement.

Algorithm 3: Compute Shadow Tint in Dimples

Input: Variables from **Algorithm 2**; `base_color` of the golf ball
Output: Shadow tint color `shadow_tint`

```
// Darken colour by 25%
shadow_amount ← 1.0 - 0.25 × best_falloff;
shadow_tint ← base_color × shadow_amount;
```

items

3.3 Lighting

An environment map [11] was used to provide ambient lighting and natural reflections. However, to offset the harsh shadow lines produced by the environment map alone, additional sphere and disk lights were blended into the scene [5]. This combination softened the shadows and enhanced overall light balance. Different lighting setups were required for the two renders (Figure 1). Without light linking, all lights influenced every object, necessitating careful adaptation for Figure 1b with multiple instances of the golf ball.

4 Dirt and Wear

4.1 Procedural Dirt Distribution

To simulate natural dirt accumulation on a golf ball, a used ball (Figure 7) was used as reference. In the reference, dirt was observed to have accumulated around the rims of the dimples in radial regions around the ball. There are also specks of dirt in random places around the ball.

The OSL shader (Algorithm 4) combined multiple techniques to procedurally replicate the dirt pattern:

Figure 7: A photograph of a used golf ball showing dirt accumulation

- **Dimple Rim Dirt:** *Simplex* [1, 8] noise-based masks were applied to target the outer edges of dimples, concentrating dirt along the rims. Three dirt centers coordinates distributed around the ball were predefined to add localized patches.
- **Random Speckles:** A high-frequency noise layer introduced scattered speckles of dirt across the surface. This was combined with a *Perlin* [8] macro noise layer to break uniformity.

4.2 Colour Variation and Realism

To mimic the varied dirt tones in Figure 7, three shades of brown were blended procedurally. Layered noise was used to mix these tones subtly across the surface.

4.3 Matte Dirt Implementation

The dirt regions were made matte by adjusting `roughnessMod` and `specularMod` (Algorithm 4) [7]. Dirt areas exhibited higher roughness and reduced specular reflection compared to the base plastic material. This created higher roughness and reduced specular reflection, highlighting contrast between the worn and glossy areas.

5 Procedural Grass Generation

A Python script was written to procedurally generate grass geometry, using RenderMan's `Curves` primitives (Algorithm 5) [6]. Each blade of grass was randomly parameterized for height, width, and bending, producing natural variation. The script outputs RIB files for modular integration into the scene.

5.1 Optimized Patch-Based Generation

To manage memory usage and improve performance, the grass was generated in a patch-based system. The scene was divided into 2×2 spatial patches, each written to a separate RIB file (Figure 8). This modular approach not only allowed for better organization but also optimized performance. Smaller RIB files reduced memory overhead, as each patch could be loaded and processed independently rather than requiring all grass blades in memory simultaneously. This setup also improved render performance, as RenderMan could efficiently cull or cache patches outside the camera's view, avoiding unnecessary computations for distant or occluded grass.

Algorithm 4: Compute Procedural Dirt & Wear

Input: Variables from **Algorithm 2** and **Algorithm 3**
Output: Updated `shadow_tint`, `roughness_mod`, `specular_mod`, `clearcoat_color`

```
outer_region ← smoothstep(0.55, 0.6, 1 - best_falloff);
dirt_noise ← high-frequency Simplex noise at pos;
dirt_location ← 0;
dirt_centers[3] ← 3 (x, y, z) points around the ball
for each dirt_center do
    compute distance to pos;
    apply jitter to radius using Perlin noise;
    compute patch_mask based on distance and jitter;
    dirt_location ← max(dirt_location, patch_mask);
dirt_mask ← clamp(outer_region × dirt_noise × patch_mask);
speckle_mask ← smoothstep(0.45, 0.6, high-frequency Simplex noise);
full_dirt_mask ← clamp(dirt_mask + (speckle_mask × (1 - dirt_mask)));
varied_dirt ← blend dirt colors using Perlin noise;
shadow_tint ← mix(shadow_tint, varied_dirt, full_dirt_mask);
roughness_mod ← mix(0.08, 1.0, full_dirt_mask);
specular_mod ← 1.0 - full_dirt_mask;
clearcoat_color ← color(0.015) × specular_mod;
```

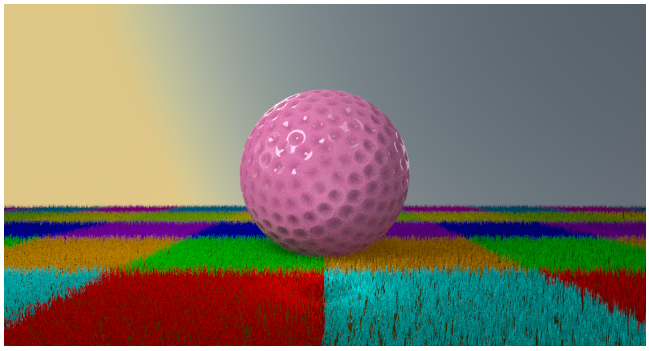

Figure 8: Debug render showing 2x2 grass patches

5.2 Optimized Density Based on Camera Distance

A distance-based density falloff was implemented to reduce computational load. The number of blades per patch was calculated dynamically based on the patch's distance from the camera. Patches closer to the camera had higher blade densities, while those farther away contained fewer blades (Figure 9).

6 Multiple Instances of Golf Balls

For the second render image (Figure 1b), a procedural approach was used to generate multiple golf ball instances with varied colours and rotations. A Python script was written to output a RIB file defining each ball instance, with predetermined positions and colours

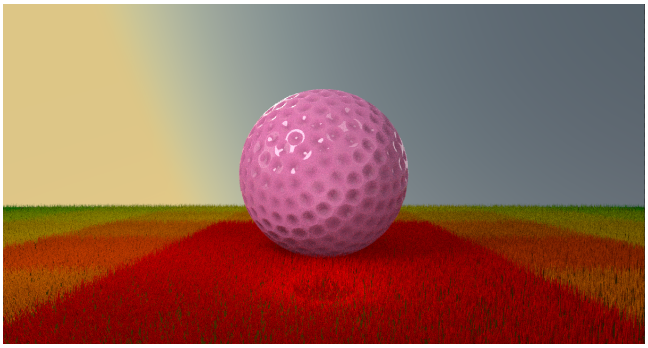

Figure 9: Render showing density fall off. Red regions indicate high density grass while green regions indicate lower density.

Algorithm 5: Procedural Grass Patch Generation

Input: Scene bounds (`x_range`, `z_range`); `patch_size`; `blade_height` range; `blade_density`
Output: RIB files containing grass patches
Clear existing grass patches directory Divide scene into a grid of patches based on `patch_size`;
for *each* `patch` **do**
  compute `patch_center` and `distance` from camera;
  determine `blade_count` for this patch based on `distance` and `density`
  write `grass_material` properties to RIB file;
  for *each* `blade` *in this* `patch` **do**
    randomize `blade_position` within patch bounds;
    randomize `blade_height` within `height_range`;
    randomize `blade_bend`;
    define `control_points` and `widths` for cubic curve;
    write `blade` curve data to RIB file;
Write master RIB file with `ReadArchive` calls for each `patch`

(Algorithm 6). Three balls were randomly rotated to create a sense of natural variation. The pink ball, serving as a focal point, was kept unrotated for visual consistency with the first render (Figure 1a). The procedural setup enabled quick iteration and reproducibility, making it an effective approach for populating the scene with multiple instances.

7 Procedural Pipeline Overview

Before concluding, it is worth noting the code's structure. The project began with manual `RIB` files to experiment with RenderMan's syntax and features. As complexity increased, `Python` scripts were used to dynamically generate `RIB` files, owing to pre-established familiarity with the `RIB` syntax. A single `render_pipeline.py` script was created to automate scene setup and rendering with one

Algorithm 6: Procedural Multi-Ball Placement

Input: `ball_list` with `ball_names`, positions, and `base_colors`
Output: RIB file with multiple ball instances
for *each* `ball` *in* `ball_list` **do**
  set `base_position (x, z)` and `color (r, g, b)`;
  if `ball_name` *is not "Pink"* **then**
    randomize rotations `(rot_x, rot_y, rot_z)`;
  apply translation and rotation transformations;
  define dimple pattern with `base_color`;
  apply displacement shader;
  assign `PxrSurface Bxdf` with dimple outputs;
  render as a sphere;

command. This procedural approach efficiently managed multiple scene variations, including lighting, grass, shading, and camera angles. The code is available at https://github.com/AnuKritiW/renderman-golfball.

8 Conclusion

This project demonstrated procedural modeling, shading, and scene composition using RenderMan RIB, Python, and OSL. A plausible dimple pattern was achieved with the Fibonacci sphere algorithm. Shading realism was enhanced through shadow tinting and procedural dirt. Grass generation was optimized with patch-based layouts and distance-based density falloff. Procedural instancing created varied multi-ball scenes. The final renders — featuring a close-up and a multi-ball scene — demonstrate that procedural techniques can produce a cohesive, visually engaging RenderMan workflow..

References

[1] Michel Anders. 2013. Simplex Noise for Open Shading Language. Retrieved May 13, 2025 from https://blog.michelanders.nl/2013/02/simplex-noise-for-open-shading-language_17.html?utm_source=chatgpt.com
[2] Martin Fasani. 2025. Evenly distributing points on a sphere. Retrieved April 29, 2025 from https://extremelearning.com.au/evenly-distributing-points-on-a-sphere/
[3] George Hart. 2013. Mathematical Impressions: Goldberg Polyhedra. Retrieved May 7, 2025 from https://www.simonsfoundation.org/2013/06/18/mathematical-impressions-goldberg-polyhedra/
[4] Jon Macey. 2025. NCCA Renderman. Retrieved April 29, 2025 from https://github.com/NCCA/Renderman
[5] Pixar. 2025. Lighting. Retrieved May 9, 2025 from Renderman26Docs-https://rmanwiki-26.pixar.com/space/REN26/19661727/Lighting
[6] Pixar. 2025. Renderman 26 Docs - Curves. Retrieved May 12, 2025 from https://rmanwiki-26.pixar.com/space/REN26/19661233/Curves
[7] Pixar. 2025. Renderman 26 Docs - Pixar Surface Materials. Retrieved May 10, 2025 from https://rmanwiki-26.pixar.com/space/REN26/19661411/Pixar+Surface+Materials
[8] Mitch Prater. 2022. OSL Shaders for RenderMan. In *ACM SIGGRAPH 2022 Educator's Forum* (Vancouver, BC, Canada) *(SIGGRAPH '22)*. Association for Computing Machinery, New York, NY, USA, Article 13, 100 pages. doi:10.1145/3532724.3535604
[9] Jack Purkiss. 2025. Renderman Ball Project. Retrieved April 29, 2025 from https://github.com/jack3761/RendermanBallProject
[10] Tom Ruen. 2025. Goldberg polyhedron 5,1 image. Retrieved May 7, 2025 from https://commons.wikimedia.org/wiki/File:Goldberg_polyhedron_5_1.png
[11] Dimitrios Savva and Jarod Guest. 2025. Limpopo Golf Course. Retrieved May 21, 2025 from https://polyhaven.com/a/limpopo_golf_course
[12] Ian Stephenson. 2007. *Essential RenderMan®*. Springer Science & Business Media.