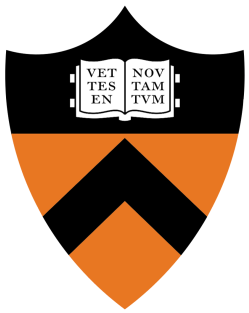

Princeton University

COS 426
Computer Graphics
December 2023

Final Project

Stamatis Alexandropoulos
sa6924@princeton.edu
Siyang Wu
sw2776@princeton.edu

1 Infinigen

1.1 Abstract

Infinigen is a revolutionary tool for creating 3D scenes of the natural world. It operates entirely through procedural generation, meaning every element, from the smallest detail to the overall composition, is constructed from scratch using randomized mathematical rules. This eliminates the need for external assets, allowing for infinite variation and customization. Infinigen's capabilities extend to generating diverse objects and scenes, including plants, animals, terrains, and even natural phenomena like fire, clouds, rain, and snow. In this project, we developed our own winter scene divided by a frozen river and complemented by the charm of snowmen and the falling snow.

1.2 Introduction

The adoption of synthetic data generated through conventional computer graphics has been on the rise in computer vision [14, 12, 15, 16, 17, 9]. Offering the advantage of unlimited quantity and automatic generation of high-quality 3D ground truth, synthetic data enable large-scale training of computer vision models and embodied agents. It is worth mentioning that numerous state-of-the-art models [19, 13] have been trained exclusively in simulated environments and exhibit remarkably strong real-world performance, even in zero-shot scenarios.

Infinigen [18] is a procedural generator of diverse photorealistic 3D scenes. Procedural generation is a promising approach for synthetic data creation that utilizes mathematical rules to generate 3D objects and scenes, employing randomized parameters for infinite variations, eliminating the need for external assets. Thus, all elements within the 3D environment, including shapes, materials, and structures, are generated algorithmically. The comprehensive procedural nature of Infinigen empowers users to customize and control every aspect of the 3D scene, ranging from individual object details to their overall arrangement, by simply adjusting the underlying mathematical rules. This results in a more diverse and dynamic set of 3D scenes.

Infinigen is built on top of Blender [8], a free and open-source graphics system that offers a variety of useful primitives for procedural generation. It contains tools for generating synthetic images and obtaining standard ground truth labels such as depth, occlusion boundaries, surface normals, optical flow, object category, bounding boxes, and instance segmentation. Additionally, the toolkit features a transpiler that automatically converts Blender node graphs into Python code.

In this project, we aim to expand Infinigen by introducing our own winter landscape.

Infinigen already incorporates a methodology where the initial step involves generating a foundational tile using either the A.N.T. Landscape Add-on in Blender or a function from FastNoise Lite [4]. This tile, possessing finite dimensions, enables the simulation of various natural processes, including erosion facilitated by SoilMachine and snowfall implemented through a diffusion algorithm in Landlab [11, 10]. The resulting tiles represent a diverse array of terrains, such as mountains, rivers, cliffs, and icebergs. The combination of these tiles can lead to a wide variety of scenes. However, instead of utilizing Infinigen's existing methods, we develop our own algorithm for creating a winter scene with a frozen river from scratch, introducing a novel and distinctive content to Infinigen. This terrain is procedurally generated by a two-stage method, where in the first stage we draw a river on the plane using B-Spline curves, and in the second stage we generate the banks of the river using a flood-fill algorithm which ensures that the landscape is smooth and is lower when close to the river. We also use several parameters to ensure that the user can control the geometry, including the curvature, width, and direction of the river, and demonstrate this by generating terrains of various shapes. To enhance the realism of the scene, we took the extra step of designing our own snowmen, new materials and implementing a custom snowfall feature. Additionally, we crafted a script that dynamically places the snowmen in random positions on the ground, contributing to the visual richness of the winter landscape.

In summary, our contribution involves an addition to Infinigen by developing a novel algorithm for scene generation, introducing unique assets and materials, ultimately providing users with a more diverse and engaging set of tools for creating winter landscapes.

1.3 Methodology

Everything in our project is procedural, from shape to texture, from macro structures to micro details, without relying on any external asset, just like Infinigen. While an artist may manually craft the form of a specific asset based on visual judgement, a procedural system generates an endless variety of items by encoding their structure and growth using generalized rules.

1.4 Terrain Generation

In this project, we developed a method to procedurally generate a terrain which contains a river and the banks of it. To generate such an uneven terrain, our method splits the plane into an $H \times W$ grid, and computes a height map $h(x, y)$ for every cell (x, y) in the grid. And to generate the height map $h(x, y)$ (For simplicity, we consider generating this function for $x, y \in [0, 1)$. In our implementation, we just scale this function to $x \in [0, H), y \in [0, W)$.), we use the following two-stage method:

Stage 1. Generate a mask of whether a cell is river or bank. In this stage, we generate a river mask function $r(x, y)$ for every cell, where $r(x, y) = 1$ if the cell (x, y) belongs to the river, and $r(x, y) = 0$ otherwise. Because the river can be regarded as a curve of certain width on the plane, our method generates $r(x, y)$ by drawing such a curve.

The problem of drawing curves reminded us of the topic of "Parametric Curves" taught in class, and we use the method of first randomly generating several control points on the plane, and then draw a Cubic B-Spline curve[5] based on these control points. As the process of drawing B-Spline curve based on control points is quite standard, in the following, we focus on introducing the method we use to generate control points.

The control points $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{N_{ctrl}}, y_{N_{ctrl}})\}$ are generated by $x_i = i \cdot \Delta_x + x_0$ and $y_i = f^R(x_i) + n_i$ for some function $f^R(x_i)$, noise term n_i, and hyperparameters Δ_x, x_0, and N_{ctrl}. We want the river that we generate meanders a lot and the curvature can vary when the user selects different hyperparameters. To do so, we choose $f^R(x) = \sin(x)/x$ as the function. And for the noise term, we set $n_{-1} = 0$, and $n_i = n_{i-1} + z$ where z is a random variable uniformly sampled from $[-n^R, n^R]$. After generating these points, to ensure y_i lies in the range of the grid, we rescale y_i into $\bar{y}_i = \frac{y_i - \min_i y_i}{\max_i y_i - \min_i y_i}$.

Based on these control points, a curve of the river is defined. To draw such curve on the plane, we sample points $B(t_i) = (\bar{x}_i, \bar{y}_i)$ for $\{t_0 = 0, t_1, \ldots, t_{N_{B-Spline}} = 1\}$ of each segment of the cubic B-Spline curve. However, this is not enough, as the river needs to have some width. To add width to the curve, we mark all cells in the region bounded by the box of $(\bar{x}_i, \bar{y}_i)$ and $(\bar{x}_i + w_i, \bar{y}_i + w_i)$ as river cells, where w_i denotes the width at point $(\bar{x}_i, \bar{y}_i)$. To ensure that the width smoothly changes, we let $w_i = CLIP(w_{i-1} + z, [W_{min}, W_{max}])$ where z is a random variable sampled from normal distribution $N(\mu = 0, \sigma = W_\sigma)$ for some hyperparameter W_σ, and then clip the sum of this random variable and previous width into range $[W_{min}, W_{max}]$ which is specified by hyperparameters W_{min}, W_{max}.

Stage 2. Generate height map. The height of the river is the consistent. Without loss of generality, we set $h(x, y)$ for cells (x, y) that belong to the river, i.e. $r(x, y) = 1$. For the rest of the cells, we want the whole terrain follows the phenomena of that the height of the bank gradually increases when the cell is farther from the river. To do so, we adopt the flood-fill algorithm[20].

More formally, we maintain a heap of all cells paired with their height, in the ascending order of the height, $H = \{(h(x, y), (x, y))\}_{(x,y)}$. Initially, H contains all cells belonging to the river, i.e. $H = \{(h(x, y), (x, y)) | r(x, y) = 1\}$. Then at each step, we pop the top of the heap $(h(x_0, y_0), (x_0, y_0))$ from H, iterate through all neighboring cells (x, y) such that $x_0 - 1 \leq x \leq x_0 + 1$ and $y_0 - 1 \leq y \leq y_0 + 1$ and $(h', (x, y))$ has not been pushed to H before for all h' (i.e. the height of cell (x, y) has not been assigned), set $h(x, y) = h(x_0, y_0) + z$ and push $(h(x, y), (x, y))$ to H, where z is a random variable uniformly sampled

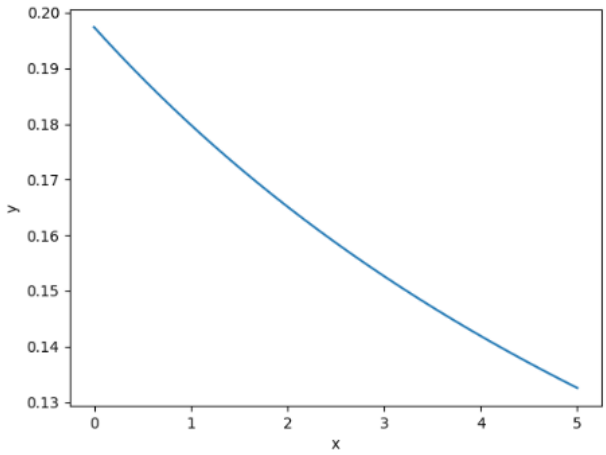

Figure 1: plot of $y = \frac{\pi}{2} - \arctan(0.1x + 5)$

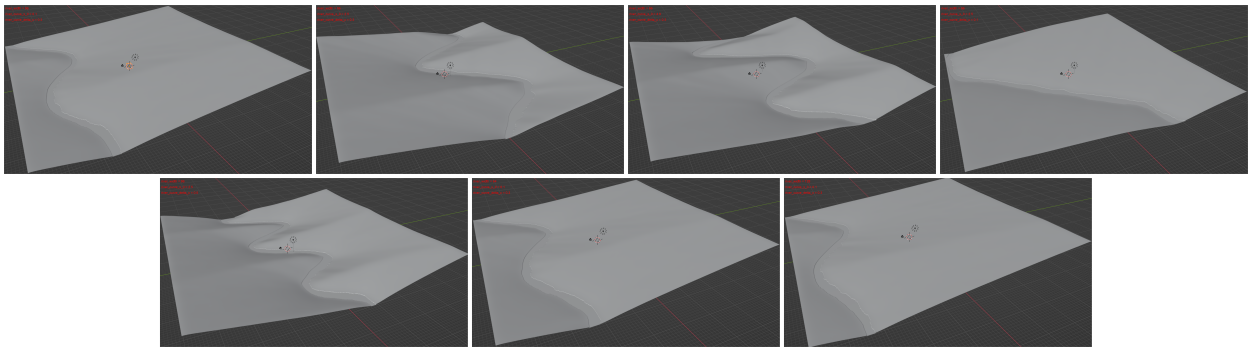

Figure 2: Terrains generated under different parameters, where "river_width" represents w_{-1}, "river_curve_x_0" represents x_0, and "river_curve_delta_x" represents Δ_x.

from $[g(h(x_0, y_0)), g(h(x_0, y_0)) + \Delta_h]$ for some function $g(h)$. We want the height of bank to increases fast when the cell is near the river, and slow when it is far. In other words, we want $g(h)$ to be large when h is small, and $g(h)$ to be small when h is large. So we set $g(h) = \pi/2 - \arctan(H_K h + H_B)$ for hyperparameters H_K and H_B. A plot of this function for $H_k = 0.1$ and $H_B = 5$ is shown in Fig. 1.

Finally, after h is generated in the above-mentioned flood-fill algorithm, we use a Gaussian blurr filter of G_σ to smooth the height map, and then set $h(x, y) = 0$ for $r(x, y) = 1$.

H	W	x_0	N_{ctrl}	n^R	$N_{B-Spline}$	W_{min}	W_{max}	W_σ	H_K	H_B	G_σ	Δ_h
2048	2048	-0.5	64	0.01	4096	$\frac{32}{2048}$	$\frac{64}{2048}$	$\frac{0.2}{2048}$	0.1	5	$\frac{32}{2048}$	0.5

Table 1: parameters for controlling the generation of terrain that we didn't enumerate

When we are generating our scene, parameters are set as Table 1 and enumerate different values for, $w_{-1} = 64, 96, 128$ (river width without any noise), $\Delta_x = 0.1, 0.3, 0.5$, and $x_0 = 0.1, 2.5, 4.9$. Results are shown in Fig. 2.

After these two stages, the height map of the terrain is generated. Then we apply the

snow and ice material, discussed in 1.4.3, to the bank and river.

1.4.1 Snowman

After the terrain is generated, we generate multiple snowmans of different geometries and materials on the field.

Each snowman consists of the following parts: body, arms, nose, eyes, and a hat. The body of a snowman is a stack of three spheres: bottom, middle, and top, of descending sizes. To generate such three spheres, we first generate a sphere of radius R_{base}. Then rescale them by following scales (here x and y-axes are the same direction as the two axes of the grid, and z-axes is the direction perpendicular to these two vectors and pointing to the sky):

Bottom: Scale along z-axis by 1, and xy-axes by $R_{base-xy}$.

Middle: Scale along z-axis by R_{mid}, and xy-axes by $R_{mid} \times R_{mid-xy}$.

Top: Scale along z-axis by R_{top}, and xy-axes by R_{top}.

Note that here we scale spheres differently along xy- and z- directions to make some sphere flatter. These spheres are stacked at the same xy positions, with $z_{base} = 0$, (z-coordinate of the center of the bottom sphere, we use the same notation for middle and top) $z_{mid} = 0.8R_{base}(1 + R_{mid})$, and $z_{top} = z_{mid} + 0.8(R_{mid} + R_{top})$. After all these steps, the body is generated.

Then we generate the arms of the snowman by two cubes of fixed position and orientation relative to the center of the middle sphere, and generate the nose, which is a cone, eyes, which are two ellipsoid, and a hat, which is a combination of two cylinders, at fixed orientation relative to the center of the top sphere. The positions for the nose, eyes, and the height relative to the center of the top sphere is a fixed function that is only relevant to top sphere radius.

In this way, we can generate the geometry of a single snowman in the scene. After that, we apply material to different part of the snowman as described in 1.4.3.

R_{base}	$R_{base-xy}$	R_{mid}	R_{mid-xy}	R_{top}
$[0.9, 1.0]$	$[1.0, 1.2]$	$[0.65, 0.75]$	$[1.0, 1.1]$	$[0.45, 0.55]$

Table 2: parameters for controlling the generation of terrain that we didn't enumerate

We use this procedure to randomly generate $N_{snowman}$ snowmans of various geometry and material by uniformly sampling the parameters in the range given in Table. 2. Then these snowmen are placed on the terrain by following steps:

- First, we generate the cell (x, y) that the center of each snowman lies in. We achieve this by generating (x, y) one by one. For the i-th snowman ($1 \leq i \leq N_{snowman}$, we repeatedly sample $x \in [D_{border}, H - D_{border}]$ and $y \in [D_{border}, W - D_{border}]$, where D_{border} is a hyperparameter that controls the distance between the snowman and the border of the terrain, until the following two criteria are met:
  - (x, y) is at least of distance D_{river} from the river, and
  - (x, y) is at least of distance $D_{snowman}$ from the center of snowman $j (1 \leq j < i)$ (here for distance, we use the metric of $dist((x_1, y_1), (x_2, y_2)) = \max(|x_1 - x_2|, |y_1 - y_2|)$)
- Secondly, the snowman i is positioned at $(x_i, y_i, h(x_i, y_i) + z_i)$, where z_i is uniformly sampled in $[0.3, 0.6]$.
- Finally, we randomly rotate the snowman by r_x, r_y, r_z degrees along x,y,z-axes, where r_x and r_y are variables uniformly sampled from $[-5, 5]$ and r_z is uniformly sampled from $[0, 360)$.

1.4.2 Snowfall

We build our own procedural snowfall based on ideas shown in [7, 6]. Our approach is totally different to Infinigen's one which is generated by a diffusion algorithm in Landlab. In our approach we use Blender's geometry nodes. We begin by creating a new group in the geometry editor and connecting a procedural geometry to it. We then add an *Instances on Points* node, using an *IcoSphere* as the instance and adjust the radius to create small snowflakes on the surface of the grid. Our snow material implementation is described below in the next subsection. To animate the snowfall, we set the position of the instances to move downwards using a *Set Position* node and a value node that counts up with the frame. By using a *Math divide* and a *Random value* node we adjust the speed and direction of the motion. We also animate a whirl effect by rotating the instances using a *Rotate Instances* node and change the center of the rotation to create a wind blowing effect. Finally, we randomized the rotation values using another random value node and adjusting the max and min sliders. As a result, we have an animated snowfall, the implementation of which can be found in *'infinigen/assets/snowfall.py'*. It is noteworthy that we insert two parameters that controls the density of the snowfall and the radius of the particles. In Fig. 3 we depict different frames of snowfall.

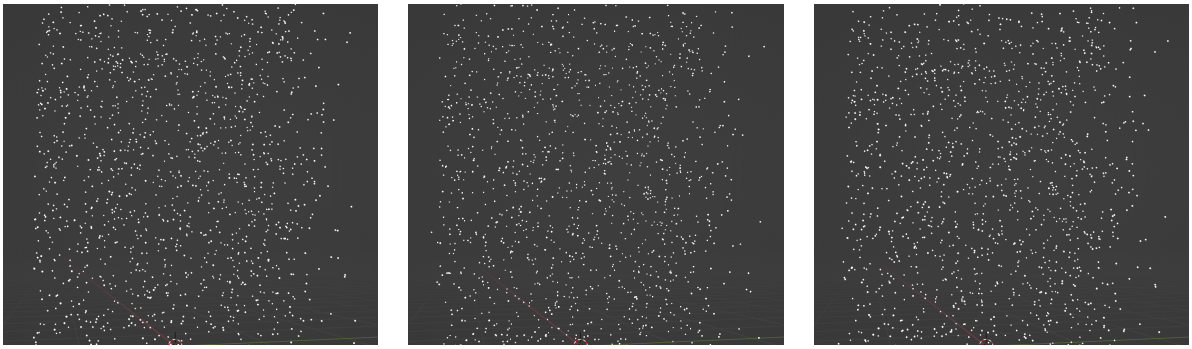

Figure 3: Different frames of snowfall animation.

1.4.3 Materials

We designed 7 new materials from scratch, an illustration of which is shown in Fig. 4. Specifically:

- Snow terrain: We create a snowy ground material using noise textures by generalizing and modifying the idea showing in [3]. First, we add a *Noise Texture* and a mix of two *Voronoi Textures* in order to create the snow with cracks. Then, we use another *Voronoi Texture* to create the rock feel and add details to it by distorting its placement using a *Noise Texture*. The texture of rocks is modified to look more natural by a parameter, $M_{snow-scale}$. This parameter affects the scale and proportion of features on the snow. The *Mix* node is then used to change the texture to a linear light, with the factor value controlled to add noise and distortion to the edges of the rocks. Another *Noise Texture* is used as a mask to decide where rocks will be placed on the snowy ground. The contrast of the noise texture is increased using a color ramp node. It is worth mentioning that we insert a parameter that controls the material's contrast, $M_{snow-contrast}$. As the parameter value increases, the snow texture takes on a progressively drier and more soiled appearance . The mask is mixed with the *Voronoi texture* in a *mix RGB* node to create a final texture that combines both the snow and rocks. The material color is finally adjusted using a *ColorRamp* node. The implementation can be found in 'infinigen/assets/material/snow_terrain.py'.

- Ice: We create an ice material using noise textures by modifying the idea showing in [2]. We begin by using a *Voronoi Texture* for the cracks, inserting a parameter, $M_{ice-scale}$, that adjusts its scale to create a more even appearance. The higher the value, the more uniform and evenly distributed the Voronoi Texture's scale becomes, resulting in a refined and cohesive appearance for the cracks. Then, we add a *Noise texture* to distort the placement of the *Voronoi Texture* for a randomized look. By mixing the *Voronoi* and *Noise* textures using a *Mix Node* we adjust the factor of the noise distortion. Af-

terward, we make the cracks smaller by changing the contrast of the *ColorRamp*. A Voronoi Texture is applied to generate a base texture, simulating a frozen surface. To enhance the realism and add intricacies, a Noise Texture is introduced to distort the placement of the Voronoi pattern. A Mix node is employed to alter the texture to a linear light, with precise control over a factor value. This adjustment introduces noise and distortion specifically to the edges of the ice cracks, enhancing the overall visual appeal. Another Noise Texture is used as a mask to decide where cracks will be placed on the frozen material. The contrast of the noise texture is increased using a color ramp node. We then create a new *Mix node* to control the colors of the cracks and ice separately. Finally, a Translucent BSDF (Bidirectional Scattering Distribution Function) is added to the material in order to enhance the realism. The Translucent BSDF is used to simulate the translucency of certain materials, allowing light to penetrate and scatter within the object, creating a semi-transparent effect. The implementation of this material can be found in 'infinigen/assets/material/ice_final.py'.

- Snowman materials: The snowman consists of multiple materials. To be more specific:
  - Snowman body: We first add a *Voronoi texture* to generate the snow's icy detail. Then we mix two *Voronoi* textures together to create contrast. After modifying the color, we increase the material's shininess and add a bump node for more realism. To enhance the detail, we duplicate the bump node, add a Noise Texture, and scale it accordingly. Additionally, we incorporate a new parameter to control the strength of the bump node, allowing for precise adjustments to the intensity of the simulated surface details. Increasing the value of the strength parameter would result in higher intensity or strength of the simulated surface details. We combine the original bump and noise textures to generate a more realistic texture. Finally, we also insert a Translucent BSDF allowing light to penetrate, creating a more shiny feeling. This material is also used for the snowfall.
  - Arms: For the arms of the snowman we develop a wooden material. Initially, we add a *Noise* and a *Musgrave* texture, and adjust their respective settings. These textures are then connected to a color ramp, which impacts the base and subsurface colors of the material. The wood grain's detail is enhanced by fine-tuning the noise scale, detail, and roughness. Distortion is also introduced to make it look more authentic, which is controlled by a new parameter. Higher values lead to more pronounced distortions, while lower values result in subtler effects.
  - Nose: For the snowman's nose we build a carrot material modifying the idea shown in [1]. First of all we use a wave texture for creating divots in the side of

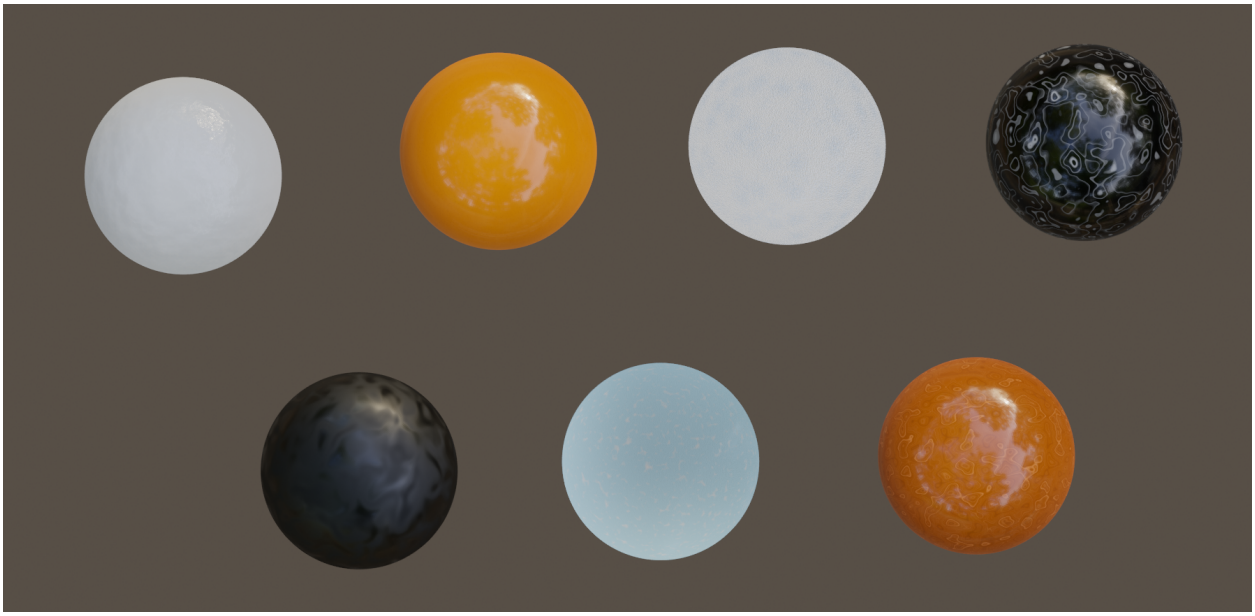

Figure 4: **Materials.** From left to right and top to bottom: snowman body, nose, snow terrain, eyes, hat, ice, hands.

the carrot. The wave texture is manipulated using various *Noise* nodes to achieve the desired randomness and contrast. Lastly, we add color to the material.

- Eyes: For the eyes, we follow a very similar approach to the arms.
- Hat: For the hat, a Noise Texture is connected to a Color Ramp node, creating a dynamic interplay of tones. Meanwhile, a separate Noise Texture is linked directly to the roughness input of the shader. This introduces fluctuations in surface roughness, influencing how light interacts with the material.

1.5 Results

The parameters that we use to generate our scene is way beyond the requirement (4), and it's impractical to enumerate every combination of these parameters. Therefore, we only demonstrate results generated by different parameters that lead to significant changes to the scene in Fig. 5.

1.6 Discussion and Conclusion

In this project, we studied how to generate a "winter holiday" scene consisting of frozen river, snow-covered uneven terrain, snowmen and snowfall. It has the following merits:

- The river on the terrain can have large curvature and meander. Moreover, we can generate various trajectories of the river by choosing different parameters.

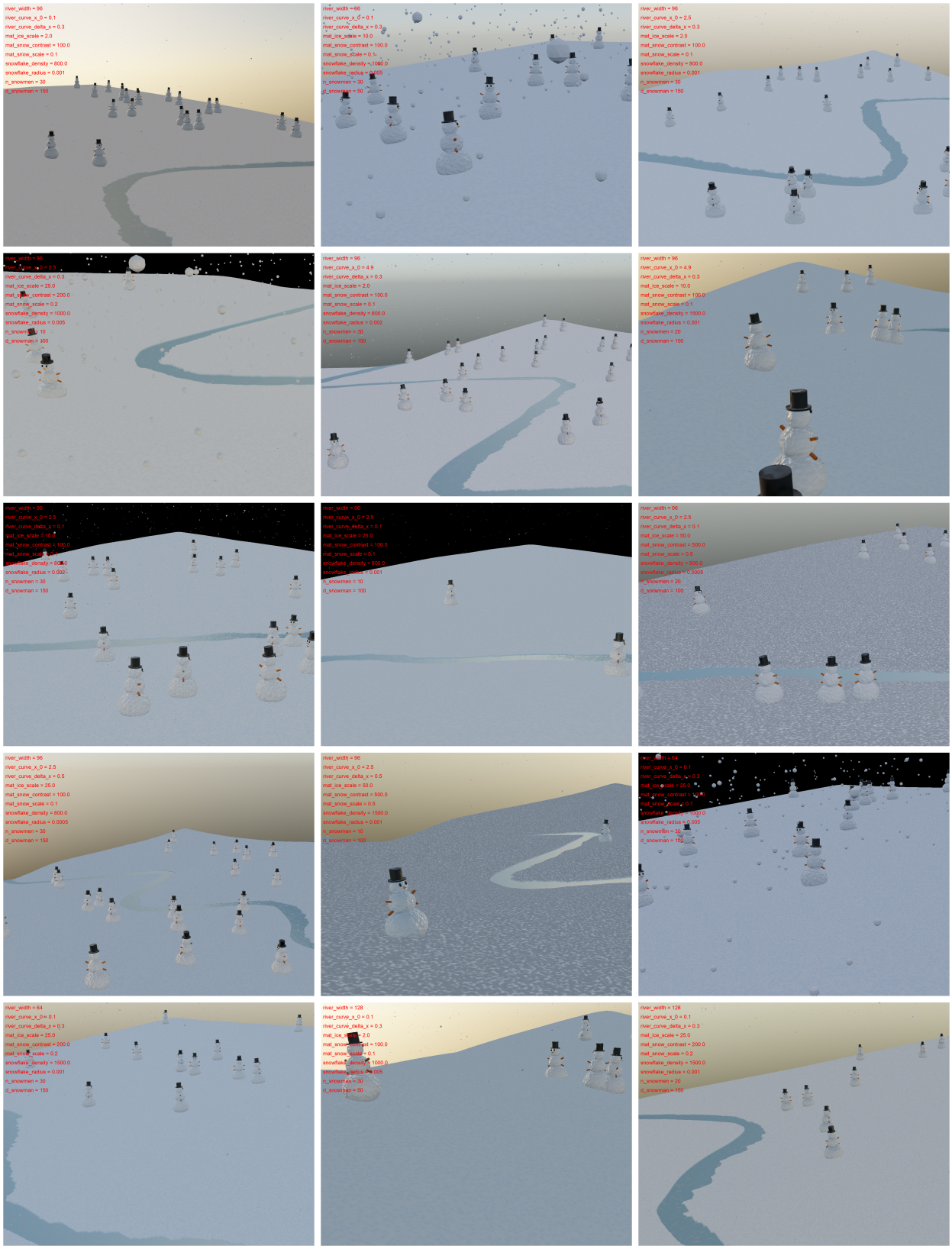

Figure 5: Scene generated by various parameter values. Parameters values are specified in top-left, where "river_width" represents w_{-1}, "river_curve_x_0" represents x_0, "river_curve_delta_x" represents Δ_x, "mat_ice_scale" represents $M_{ice-scale}$, "mat_snow_contrast" represents $M_{snow-contrast}$, and "mat_snow_scale" represents $M_{snow-scale}$.

- The slope of the bank is created by increasing the height of the landscape when it is farther from the river.
- We create the phenomena of snowfall by adding movement to snowflakes.
- We design the asset of a realistic snowman, which has many details, including the body, arms, a nose, eyes, and a hat.By introducing a detailed and realistic snowman asset, users can elevate their scenes, adding a sense of playfulness and seasonal charm.
- We designed from 7 new materials, enhancing Infinige's library providing users with a more diverse set tools.

References

[1] JRoss 3D. *Carrot Material — Blender Tutorial*.

[2] Ryan King Art. *Procedural Ice Material (Blender Tutorial)*.

[3] Ryan King Art. *Procedural Snowy Ground Material*.

[4] Auburn. *FastNoise*, 2018.

[5] Wolfgang Böhm, Gerald Farin, and Jürgen Kahmann. A survey of curve and surface methods in cagd. *Computer Aided Geometric Design*, 1(1):1–60, 1984.

[6] CGMatter. *Procedural snow in blender*.

[7] Point Cloud. *Blender Geometry Nodes Tutorial - Procedural Snowfall*.

[8] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.

[9] Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. Procthor: Large-scale embodied ai using procedural generation. *Advances in Neural Information Processing Systems*, 35:5982–5994, 2022.

[10] Daniel EJ Hobley, Jordan M Adams, Sai Siddhartha Nudurupati, Eric WH Hutton, Nicole M Gasparini, Erkan Istanbulluoglu, and Gregory E Tucker. Creative computing with landlab: an open-source toolkit for building, coupling, and exploring two-dimensional numerical models of earth-surface dynamics. *Earth Surface Dynamics*, 5(1):21–46, 2017.

[11] Eric Hutton, Katy Barnhart, Dan Hobley, Greg Tucker, Sai Nudurupati, Jordan Adams, Nicole Gasparini, Charlie Shobe, Ronda Strauch, Jenny Knuth, et al. Giuseppecipolla95. *Amanda Manaster, Langston Abby, Kristen Thyng, and Francis Rengers. landlab*, 4:6, 2020.

[12] Hei Law and Jia Deng. Label-free synthetic pretraining of object detectors. *arXiv preprint arXiv:2208.04268*, 2022.

[13] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5, 2020.

[14] Jiankun Li, Peisen Wang, Pengfei Xiong, Tao Cai, Ziwei Yan, Lei Yang, Jiangyu Liu, Haoqiang Fan, and Shuaicheng Liu. Practical stereo matching via cascaded recurrent network with adaptive correlation. In *CVPR*, 2022.

[15] Jiankun Li, Peisen Wang, Pengfei Xiong, Tao Cai, Ziwei Yan, Lei Yang, Jiangyu Liu, Haoqiang Fan, and Shuaicheng Liu. Practical stereo matching via cascaded recurrent network with adaptive correlation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16263–16272, 2022.

[16] Lahav Lipson, Zachary Teed, Ankit Goyal, and Jia Deng. Coupled iterative refinement for 6d multi-object pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6728–6737, 2022.

[17] Zeyu Ma, Zachary Teed, and Jia Deng. Multiview stereo with cascaded epipolar raft. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXI*, pages 734–750. Springer, 2022.

[18] Alexander Raistrick, Lahav Lipson, Zeyu Ma, Lingjie Mei, Mingzhe Wang, Yiming Zuo, Karhan Kayan, Hongyu Wen, Beining Han, Yihan Wang, Alejandro Newell, Hei Law, Ankit Goyal, Kaiyu Yang, and Jia Deng. Infinite photorealistic worlds using procedural generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12630–12641, 2023.

[19] Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. In *Neural Information Processing Systems*, 2021.

[20] Wikipedia. *Flood fill*, 2023.