Houdini PCG Vegetation & Props

Title: Houdini PCG Vegetation & Props
Date: Jun 2025
Role: Solo
Engine: Houdini
Demo Video: https://youtu.be/CUhrmoAulYU

Introduction

Parameterized HDAs to generate flowers, cacti, trees, rocks, a house kit, and a spline-based bridge for rapid world building.

Technical Design

Rock Generator

Introduction

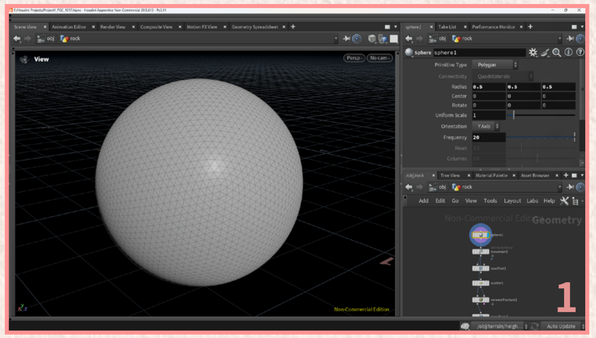
A procedural tool for generating stylized, low/high poly rocks from a single sphere input. it focuses on fast variation and clean UVs, giving controllable erosion, scale, and fracture density for use as a modular environment pieces.

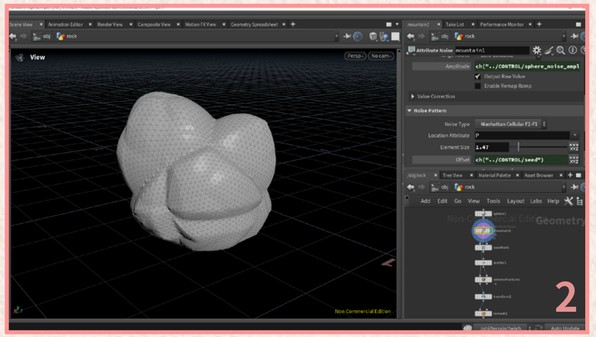
Process

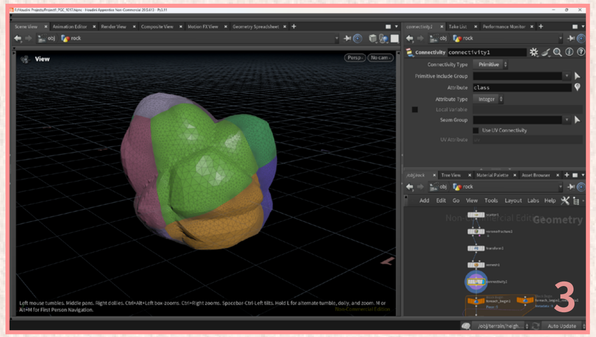
I started from a clean sphere and added noise to it. I then fractured it and applied per-piece, centroid-aware variation by Connectivity. Stable, deterministic randomization in the loop over all pieces introduced subtle size changes across chunks, breaking the realistic shape and creating a more stylized and low-poly rock structure. After this, the shape of the rock was mostly set. For a clean, ready output, I added another remesh to lower the face count, recompute normal, and project UV at the end.

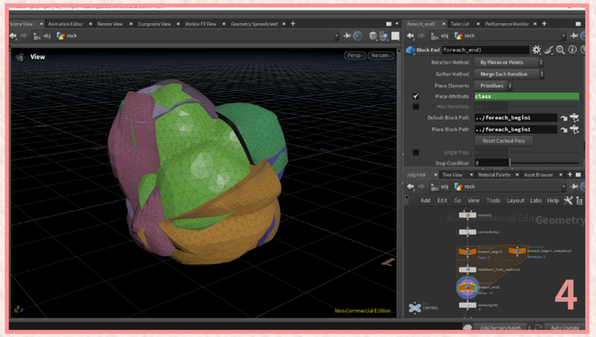
Exposed Parameters

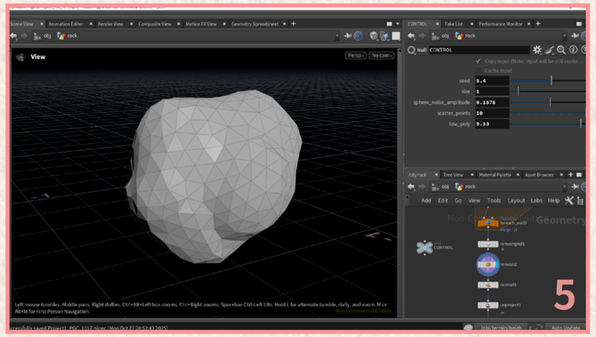
- Seed (Mountain Offset): Global variation for the micro form
- Sphere Noise Amplitude: Strength of silhouette deformation, higher = eroded look
- Scatter Points: Controls fracture granularity (piece count and size)
- Size (Transform): General scale
- Low Poly: Controls remesh element min size, higher = more stylized, low-poly shape











Technical Design

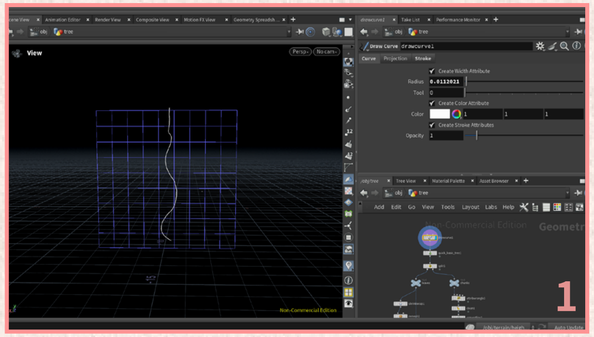
Tree Generator

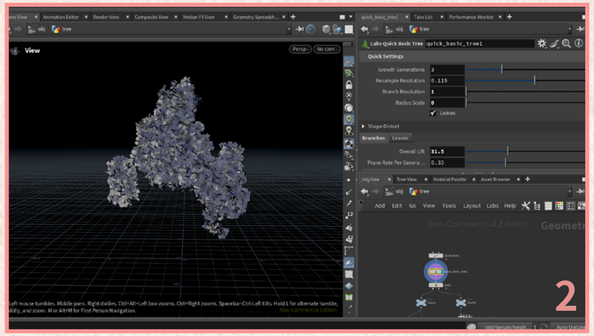
Introduction

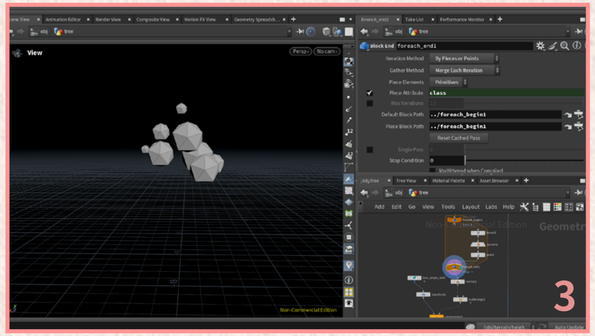
A stylized tree tool built on top of SideFX Labs' Quick Basic Tree, but heavily tuned for more graphic, readable silhouettes and foliage masses. It exposes higher-level controls for trunk flow, branch spread, and leaf clustering so you can quickly art-direct iconic trees that feel cohesive with the rest of the stylized environment.

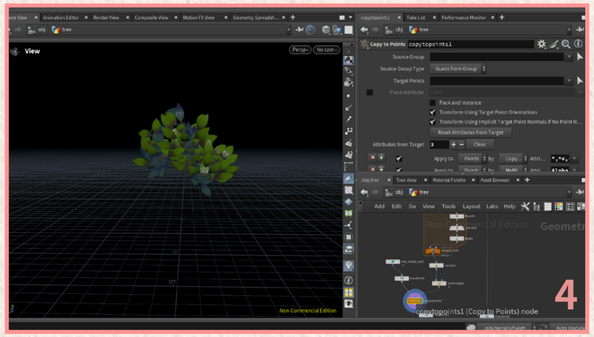
Process

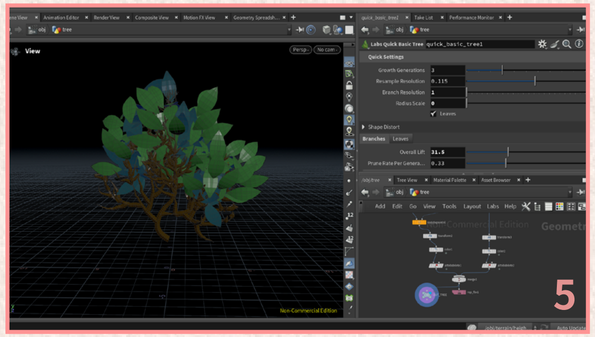
This generator is built on top of SideFX Labs' Quick Basic Tree, wrapped into a cleaner, export-focused tool. A guide curve (or default presets) feeds the trunk/branch system, where distance-based remapping, lift masks, directional noise, and pruning controls shape the main silhouette and secondary branches. Leaves are generated using Labs Tree Simple Leaf and scattered with Scatter Align, which handles orientation, density, and relaxation so foliage wraps naturally around the canopy. The result is then cleaned and merged.











Technical Design

Cactus Generator

Introduction

A parameterized cactus HDA that builds ribbed trunks and instanced spines from a single profile curve. It allows quick art-direction of silhouette, height, and spike patterns while staying export-ready for use as assets or scatters.

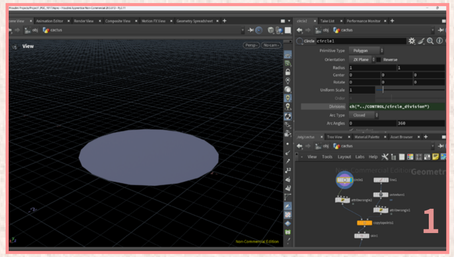
Process

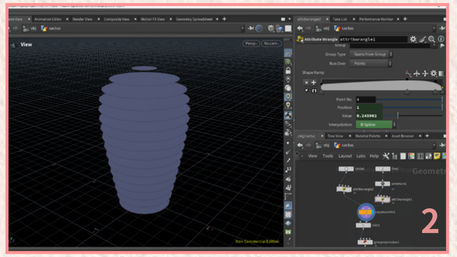
I start from a horizontal circle as the cross-section and drive its radius by UV (u) with a Shape Ramp (scaled by a global multiplier). I copy these profiles along a guideline and Skin them to form the base. Using a column_id per cross-section point lets me split odd/even columns into inner/outer rib groups. Scaling those separately creates the characteristic ribbed cactus silhouette.

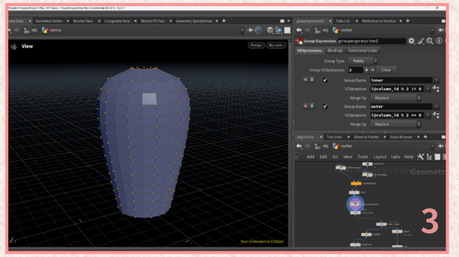
I then branch the graph into two parts:

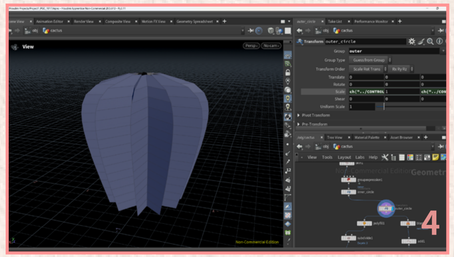
- Base: Smooth the result.
- Spines: From the outer ring, convert to a line, resample, and use Orient Along Curve for spike direction. I add angle variation (Attribute Randomize) to create clumps, then Ray the points to the base surface. Each spike is generated from a line whose width is driven by a UV-ramped @width, then Polywire builds the controllable spike mesh.

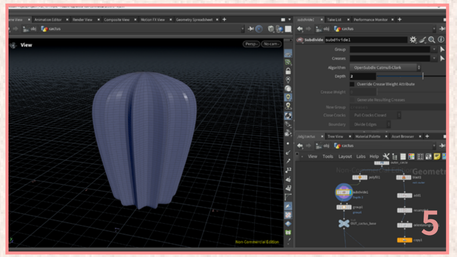
Finally, I prune extra attributes and merge the base and spines.

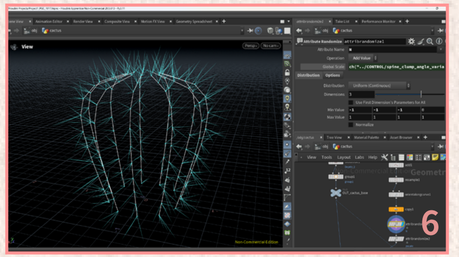


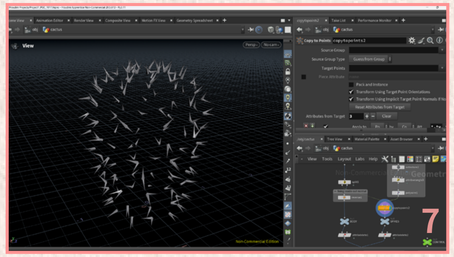


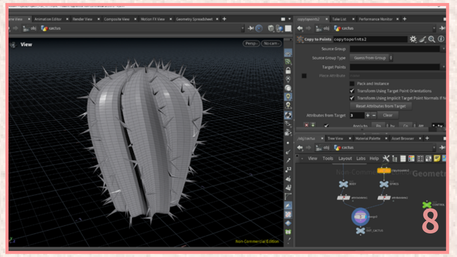












Technical Design

Cactus Generator

Exposed Parameters

[Base Cactus Shape]

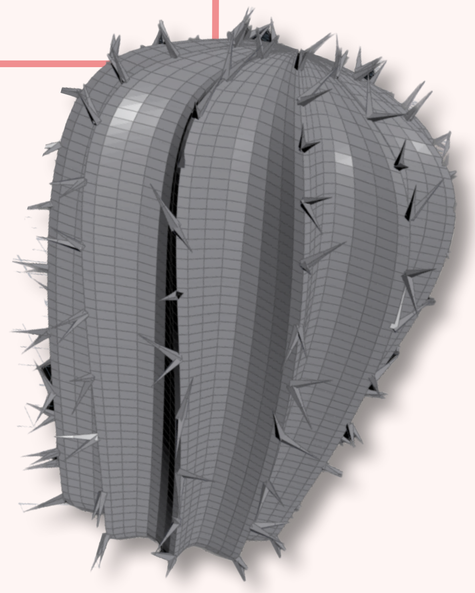
- Circle Division: Cross-section resolution, how dense the "gullies" are
- Height: Overall cactus height (guideline length)
- Shape Ramp: Radius profile along height (cross-sectional shape along up-direction)
- Inner Shape Scale: Scale for odd columns (inward rib amplitude)
- Outer Shape Scale: Scale for even columns (outward rib amplitude)
- Cactus Twist: Global helical twist

[Spikes]

- Segments: How many spikes along outer rib (dense/sparse)
- Spine Count: How many spikes on the outer ring (per sample point)
- Spine Clump Angle Variation: Random local rotation around the normal for clustered look
- Scale Clump Min/Max: Per-spike scale range for natural size variation
- Spine Division: spike smoothness
- Spine Width Ramp: Use a curve to control the shape of spikes
- Spine Length: Line length for each spike

Problem & Solution

When exporting to Unreal, I initially saw inside-out normals on the base. Inserting a Reverse after the base fixed this issue. But I wanted to locate where the root problem was and fix that. So I backtracked to the very first nodes, and found that the normals were already inside-out in the skin node.



Technical Design

Flower Generator

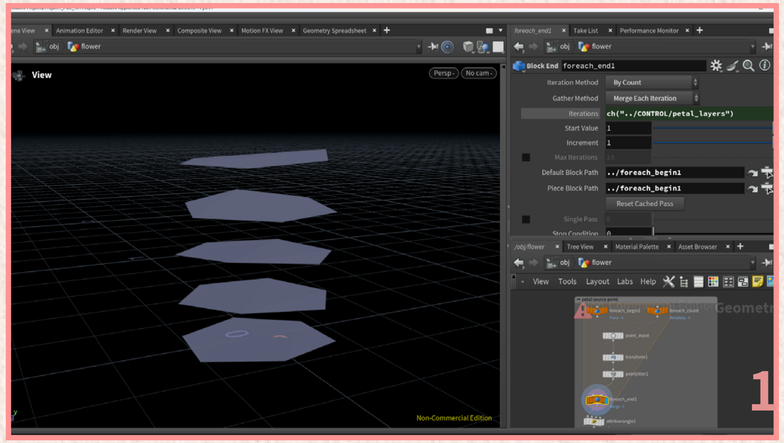
Introduction

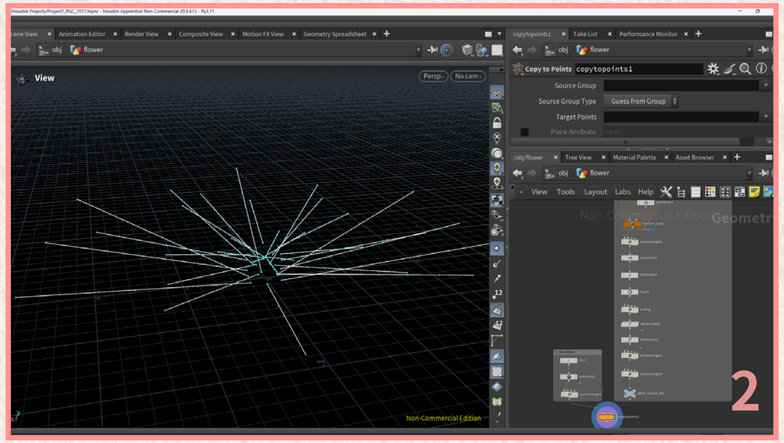
A layered flower generator that constructs petals, stamen, and subtle curl/bend from fully controllable ramps and noise. It's designed to produce cohesive flower clusters with natural variation suitable for close-up shots and large ground cover.

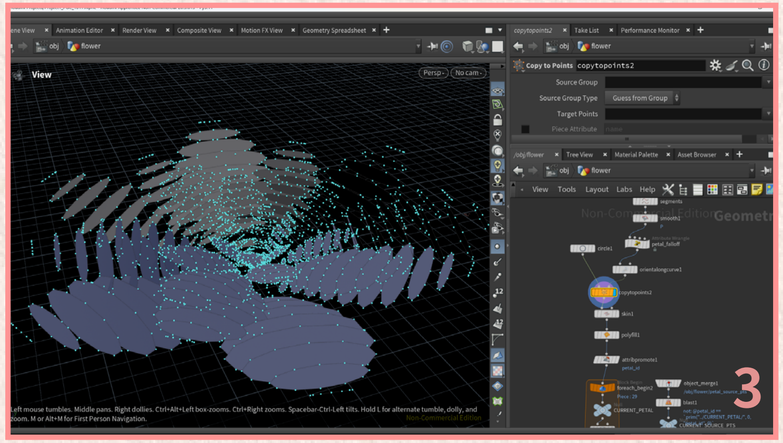
Process

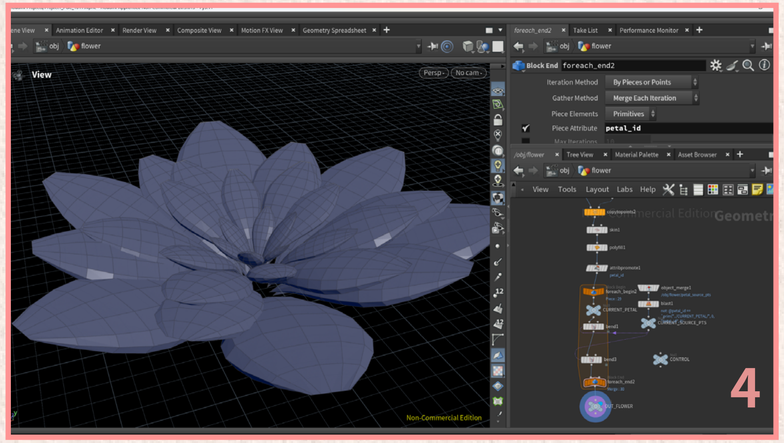
I build the flower in layers of petals. A For-Each loop scatters concentric circle profiles; each layer's center is offset by layer_spacing*iteration and given a small randomized Y-rotation and jitter. The top layer is isolated and scaled down to form the stamen.

For petal geometry, I loft circles along guide lines: a line gets UVs, and ramps drive its width/depth profile along its length. I copy the profiles to those points, Skin to form each petal sheet, fill gaps, and assign a per-petal petal_id. Finally a For-Each over petal_id applies a controllable curl/bend, then output.









Technical Design

Flower Generator

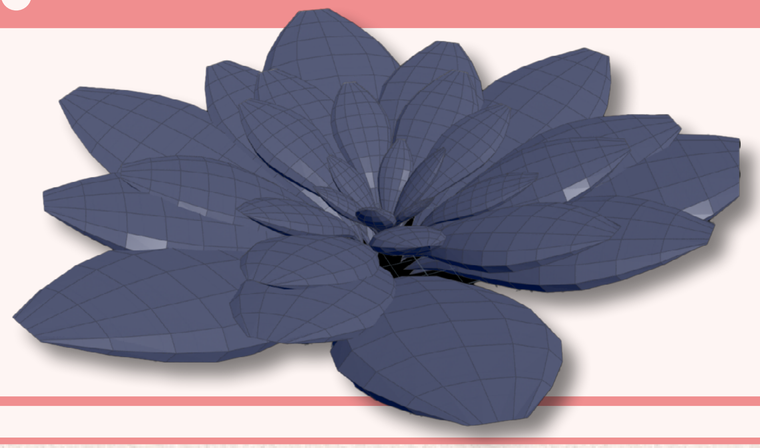
Exposed Parameters

[Petals - Layout & Layers]
- petal_count: Number of petals per layer.
- petal_layers: Number of concentric layers.
- layer_size: Uniform scale of each layer's circle profile.
- layer_spacing: Radial offset between layers.
- center_variance: Jitter of layer centers for organic asymmetry.
- variance_seed: Seed for all per-layer randomization.
- dir_random: Noise strength for direction/up variation.

[Petals - Shape & Profile]
- min_width / max_width / width_variance: Petal width range + randomness across layers.
- min_length / max_length / length_variance: Petal length range + randomness.
- min_depth / max_depth / depth_variance: Thickness (Z) range + randomness.
- line_segments: Guide-line tessellation (smoothness along root → tip).
- petal_width / petal_depth: Base multipliers for cross-section scale.
- petal_width_ramp: Width profile along the petal.
- petal_depth_ramp: Depth/thickness profile along the petal.
- stem_segments: Smoothing/tessellation for the guide lines before lofting.
- petal_falloff (ramp): Vertical falloff applied along u to sag the petal toward the stem.

[Stamen]
- top_scale: Scales the top (last) layer to form the stamen cluster.

[Curl & Bend]
- curl: Bend angle applied per petal in the For-Each (petal_id).
- bend: Additional bend amount (if separate from curl), otherwise leave.



Technical Design

House Generator

Introduction

A modular house/building generator that converts a single blockout into a finished facade with floors, windows, beams, roof shapes, and Unreal materials IDs. It standardizes proportions and detailing while staying flexible enough to support different architectural styles with different geometry inputs.

Process

I take a simple blockout (two boxes: body + roof wedge) and tag faces with Unreal material slots (s@unreal_material) so walls, roof, and roof–wall transition can be styled independently. A preprocess pass builds working groups (wall / wall_non_roof / wall_roof / roof), dissolves flat edges, and fuses roof + body into a single, watertight shell.

Roof toolchain

- Shape: Identify roof edges (roof_id) and introduce subtle imperfections via randomized edge extrusions. To avoid fragile PolyExtrude insets, I use a safe inset: derive edge normals/tangent(u/v) (Polyframe), transfer them to the shell, then P += N * gradient * amount to push lips/eaves cleanly without self-intersection.
- Curl: Build a falloff (@gradient) over step distance along the eaves and offset along N for a light "tilt" at the roof edge.
- UVs: Each roof island is UVFlattened, then I compute its "up" direction in UV space (Polyframe → dot/atan2) and apply a per-island UVTransform so textures stay aligned even if the physical roof is rotated. Texel density is normalized and UVs are transferred back.
- Extrude: extrudes the wall along different curves to achieve the shape we want.

Walls toolchain

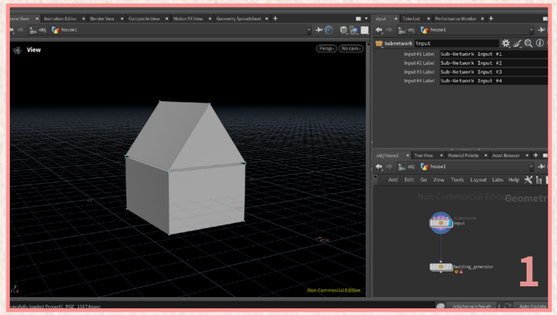
- Slice by floors → per-face loop. Each wall piece is split and divided vertically by Floor Height using a Divide SOP (sizeY bound to the control), then iterated in a foreach over primitives.
- Window lanes & spacing. A Chain SOP lays out repeat locations for windows
- Window shells & insets. Selected points/prim groups (inside_window) are transferred/promoted and inset with a negative-distance PolyExtrude to push openings inward; non-beam wall areas get a slight positive thickness for reveals.
- Sills/headers (beams). Bottom/top edges of each opening are extracted and swept
- Push windows flush. Curves/frames are oriented, a Ray SOP gets the hit distance, it's transferred, then a wrangle offsets geometry along the local up to sit perfectly on the facade.
- Basement cut. A Clip SOP uses the wall’ s bbox Y-min as origin and Basement Height control for distance

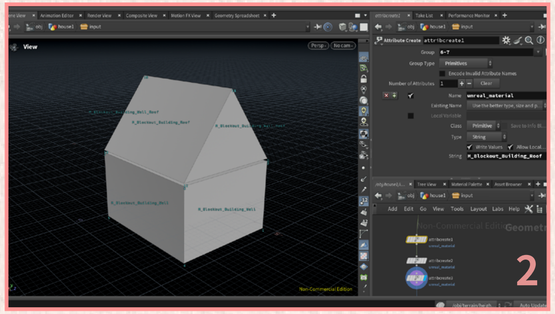
Finally, roof + walls are merged and a post pass assigns final Unreal material slots (basement/beam/window/wall/roof/roof_beam) so the asset drops into UE with correct materials out of the box.

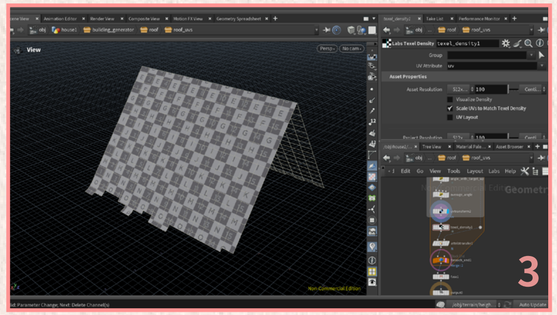
Technical Design

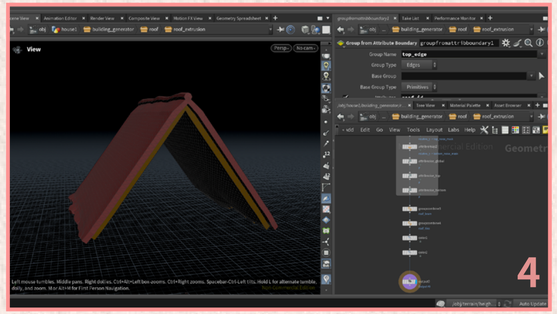
House Generator

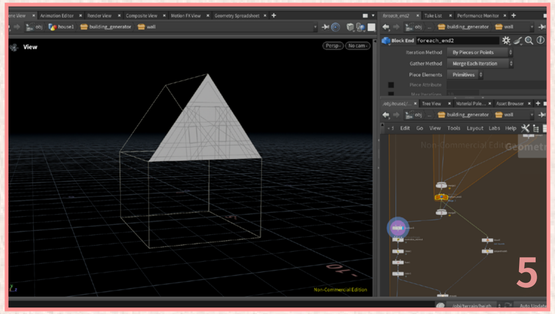
Process

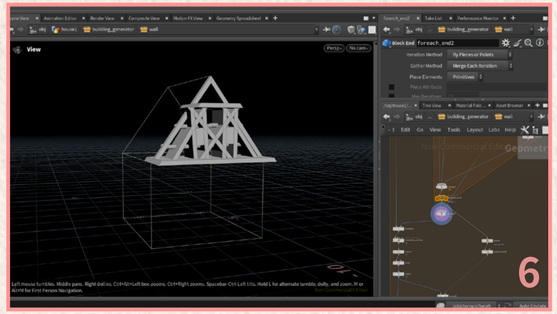


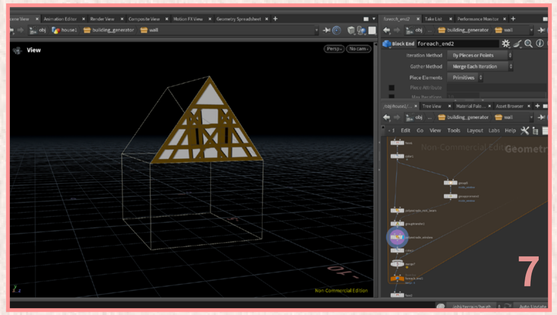


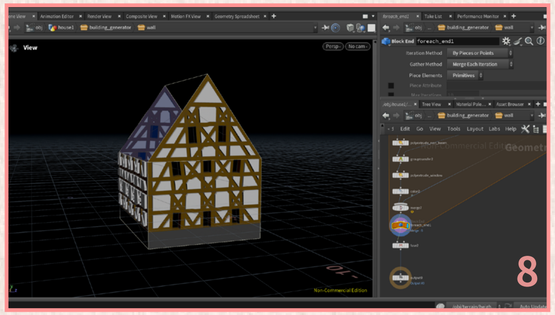


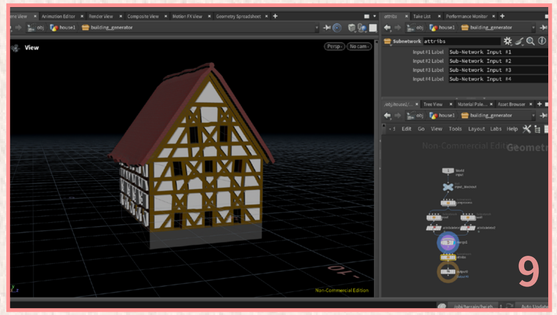












Exposed Parameters

[Global / Layout]

- Snap Distance: Grid-snaps the input blockout so divisions/extrusions land cleanly (prevents sliver faces).
- Floor Height: Vertical module used by the wall/window logic.
- Basement Height: Adds a plinth course; also used as a clamp for window placement.

[Structure / Detailing]

- Beam Thickness: Thickness for beams/frames/eaves; drives PolyExtrude/offset amounts.
- Window Height: Target opening height for generated windows.
- Min Window Dist to Wall: Minimum clearance from window edges to panel borders (prevents paper-thin margins).
- Window Spacing: Horizontal spacing between windows within a bay (regularity vs density).

[Materials (Unreal)]

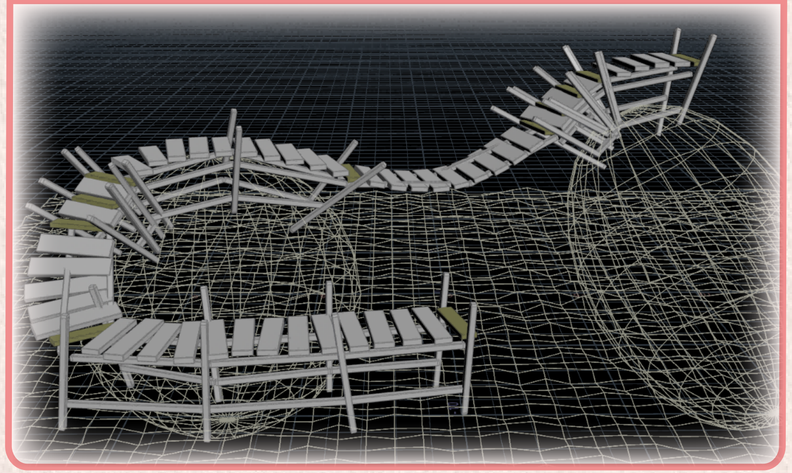
- Basement Material / Beam Material / Window Material / Wall Material / Roof Material / Roof Beam Material

Technical Design

Bridge (Spline-based) Generator

Introduction

A spline-driven bridge tool that conforms to arbitrary terrain and automatically separates hanging spans, ramps, and stairs. It places planks and support structures procedurally along the curve, creating a stylized bridge that can be reshaped by editing a single path.



Process

I start from a user-drawn curve that defines the bridge path on a random test scene. The curve is resampled and projected (Ray) onto the target landscape, then lifted slightly (Peak + Ray). From there, I build a robust frame of attributes:

- Curve framing: Convert Line + Orient Along Curve generate per-point tangent and up, and I store distance along the curve for later logic.
- Hanging & attached: A Group Expression marks points whose distance from ground exceeds a Bridge Lift threshold as Hanging. The curve is split into:
  - Hanging segments: Resampled, midpoints jittered, and re-smoothed to create a natural sag in the suspended section; tagged as hanging_bridge.
  - Attached segments: Run through a For-Each where I measure slope using dot product ot up and tangent. The flatter parts become stairs, and the steeper parts remain rigid bridge ramps.

These three paths—hanging, bridge, and stairs—are merged back into a single guide curve, resampled, and oriented again. From this unified guide I branch into: plank generation and supports generation.

[Plank Generation]

- I isolate end points vs interior points, and assign them different scales, add small noise, and merge them together.
- A Box (with bevel) is used as the plank mesh, then Copy to Points places planks along the guide using the stored tangent/up.
- Finalize with a simple distance flag, Normals, and Auto UV for tiling wood materials.

[Support Generation]

I build three main support types:

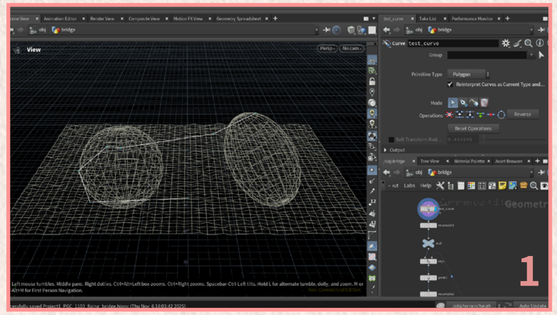
1. Vertical supports along non-hanging sections
   - Sample points along bridge curve using support_ length.
   - For each sample, build a vertical post from the plank down to the ground (Ray), and adjust based on scale and offset N so posts are evenly spaced and always meet the terrain.
2. Longitudinal supports parallel to the bridge
   - Reuse the same dir and frame to place beams that run along the curve, slightly offset from the deck.
3. Additional side/edge supports
   - Variations of the above, sharing the same data-driven frame for consistent alignment.

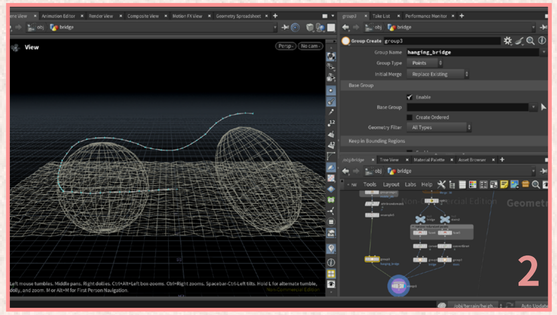
All supports are instanced from a beveled beam mesh via Copy to Points (using the curve frame), then merged with the planks into a single, game-ready bridge.

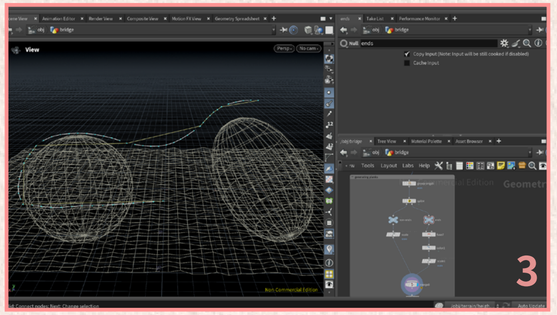
Technical Design

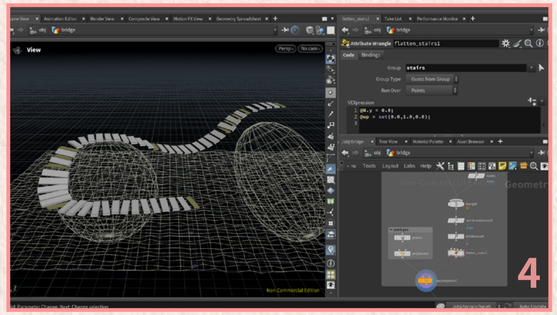
Bridge (Spline-based) Generator

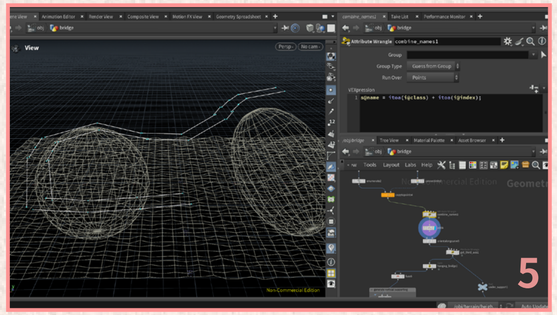
Process

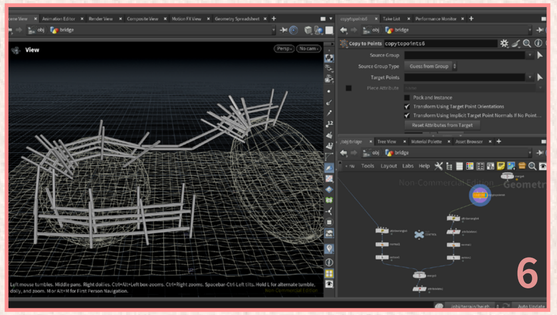


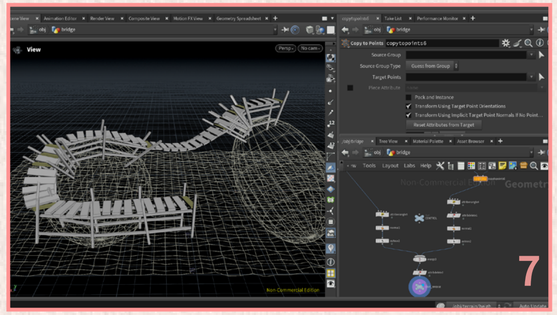












Exposed Parameters

[Curve & Section Classification]
- bridge_lift: Height threshold used to mark segments as hanging vs. on-ground.
- stair_slope_threshold: Slope cutoff; flatter than this becomes stairs, steeper remains main bridge/ramp.

[Planks]
- plank_scale: Base uniform scale for regular planks along the span.
- plank_pivot_scale: Scale for end/anchor planks to visually ground the bridge.
- scale_randomize: Amount of per-plank size variation for a hand-built feel.
- direction_randomize: Small random directional/normal variation to break perfect alignment.

[Support Sampling]
- support_length: Target spacing / segment length for placing structural supports along the curve.

[Vertical Supports]
- vertical_support_scale: Overall size multiplier for vertical posts beneath the bridge.
- vertical_support_N: Controls how many vertical supports are distributed along the span.

[Horizontal Supports — Layer 1]
- horizontal_support_scale_1: Thickness/size of the first layer of horizontal beams running parallel to the bridge.
- horizontal_support_N_1: Count/density of these beams.

[Horizontal Supports — Layer 2]
- horizontal_support_scale_2: Thickness/size of the second horizontal support layer.
- horizontal_support_N_2: Count/density of this secondary layer.