

Algorithmic Design Workbook with Houdini

アルゴリズミックデザインワークブック

Junichiro Horikawa / 堀川淳一郎

Algorithmic Design Workbook with Houdini

アルゴリズミックデザインワークブック

Junichiro Horikawa
堀川淳一郎 著

Preface はじめに

Thank you for having an interest in this book. This book is a algorithmic design work-book for Houdini contains 21 diverse exercises with two sample answers for each. This is made for Houdini beginners to intermediates who want to improve their procedural modeling skills.

本書に興味を持っていただきありがとうございます。本書は21もの課題と、それぞれの課題につき2種類のサンプルの答えを収めたHoudiniのためのアルゴリズミックデザインのワークブックです。このワークブックはHoudiniでプロシージャルモデリングを行う初級者から中級者用にそのスキルを上達させることを目的に書かれています。

Contact お問い合わせ先

email: jh@orangejellies.com

twitter: @jhorikawa_err

website: jhorikawa.com

File Download ファイルダウンロード

You can download all the hip files for answers shown in this book from following url.

この本に書かれているすべての答えのHoudiniのHIPファイルは、次のURLからダウンロードできます。

<https://github.com/jhorikawa/AlgorithmicDesignWorkbookWithHoudini>

System Requirement 動作環境

Houdini 18.0 ~ (Any License / Any Platform)

Disclaimer 免責事項

Although the author and publisher have made every effort to ensure that the information in this book was correct at press time, the author and publisher do not assume and hereby disclaim any liability to any party for any loss, damage, or disruption caused by errors or omissions, whether such errors or omissions result from negligence, accident, or any other cause.

本書は情報の提供のみを目的としております。筆者はその内容について最新の注意を払っておりますが、その内容が最新のものであるかどうか、正確なものであるかどうか等について保証するものではなく、著者はいかなる責任も負うものではありません。また本書のご利用により万一、読者様に何らかの不都合や損害が発生したとしても、筆者はいかなる責任を負うものではありません。

Table of Contents 目次

| | |
|--|----|
| Preface / はじめに | 2 |
| Contact / お問い合わせ先 | |
| File Download / ファイルダウンロード | |
| System Requirement / 動作環境 | |
| Disclaimer / 免責事項 | |
| How to use this book / 本書の使い方 | 5 |
| Houdini Exercises / Houdini練習帳 | |
| 01. Look at a target / ターゲットに向く | 6 |
| 01-A. Look directly / 直接見る | |
| 01-B. Look horizontally / 水平に見る | |
| 02. Contouring / オブジェクトをコンター割りする | 12 |
| 02-A. Contouring with loop / ループを使ってコンター割りする | |
| 02-B. Contouring simultaneously / 一気にコンター割りする | |
| 03. Random walk / ランダムに歩く | 18 |
| 03-A. Random walk with solver / ソルバーを使ってランダムに歩く | |
| 03-B. Random walk with noise / ノイズ関数を使ってランダムに歩く | |
| 04. Move along path / パスに沿って動かす | 24 |
| 04-A. Move object with Carve / Carveを利用してものを動かす | |
| 04-B. Move object with Curve uv / 曲線のUVを利用してものを動かす | |
| 05. Switch objects / 条件に応じてオブジェクトをスイッチする | 30 |
| 05-A. Switching with Stamp / Stampを利用してスイッチする | |
| 05-B. Switching with loop / ループを利用してスイッチする | |
| 06. Sloped roof / 勾配屋根を作る | 36 |
| 06-A. Create roof with loft / ロフトを使って屋根を作る | |
| 06-B. Create roof by pinching center / 中心をつまんで屋根を作る | |
| 07. Snow on object / 雪を積もらせる | 42 |
| 07-A. Piled snow / 積もった雪 | |
| 07-B. Think snow layer / 薄い雪の層 | |
| 08. Voxelize object / オブジェクトをボクセル化する | 48 |
| 08-A. Voxelize using volume / ボリュームを使ってボクセル化する | |
| 08-B. Voxelize by moving points / ポイントを移動してボクセル化する | |
| 09. Attractor transformation / アトラクターで変形する | 54 |
| 09-A. Transformation with loop / ループを使って変形する | |
| 09-B. Transformation with ramp / Rampを使って変形する | |
| 10. Variation layout on grid / グリッド上にバリエーションを配置する | 60 |
| 10-A. Layout using nested loop / 二重ループを使ってレイアウトする | |
| 10-B. Layout using attributes / アトリビュートを使ってレイアウトする | |

| | |
|---|-----|
| 11. Undulated panels / 波打つパネルを作る | 66 |
| 11-A. Place panels on flat plane / 平面の上にパネルを配置する | |
| 11-B. Place panels on any surface / 好きなサーフェスにパネルを配置する | |
| 12. Rotational slicing / オブジェクトを回転方向でスライスする | 72 |
| 12-A. Slice cylindrically / 円柱状にスライスする | |
| 12-B. Slice spherically / 球状にスライスする | |
| 13. Image remeshing / 画像を使ってリメッシュする | 78 |
| 13-A. Remeshing with voronoi / ボロノイを使ってリメッシュする | |
| 13-B. Remeshing with dual mesh / デュアルメッシュを使ってリメッシュする | |
| 14. Vector field control / オブジェクトでベクトル場を操作する | 84 |
| 14-A. 2D vector field / 二次元のベクトル場 | |
| 14-B. 3D vector field / 三次元のベクトル場 | |
| 15. Gouged surface / サーフェスを削る | 90 |
| 15-A. Gouge using Mountain / Mountainを使って削る | |
| 15-B. Gouge using voronoi / ボロノイを使って削る | |
| 16. Particle morphing / パーティクルをモーフィングする | 96 |
| 16-A. Linear morphing / 線形モーフィング | |
| 16-B. Spread morphing / 拡散モーフィング | |
| 17. Silhouette morphing / シルエットをモーフィングする | 102 |
| 17-A. Linear morphing / 線形モーフィング | |
| 17-B. Morphing with easing / イージング付きモーフィング | |
| 18. Isosurface lattice / アイソサーフェスのラティスを作る | 108 |
| 18-A. Lattice using metaball / メタボールを使ったラティス | |
| 18-B. Lattice using volume / ボリュームを使ったラティス | |
| 19. Braid rope / 三編みロープを作る | 114 |
| 19-A. Straight braid rope / 直線的な三編みロープ | |
| 19-B. Braid rope along curve / 曲線に沿った三編みロープ | |
| 20. Fractal column / フラクタルな柱を作る | 120 |
| 20-A. Fractal using feedback loop / フィードバックループを使ったフラクタル | |
| 20-B. Fractal using solver / ソルバーを使ったフラクタル | |
| 21. Surface marbling / サーフェスにマーブル模様を作る | 126 |
| 21-A. Marbling with loop / ループを使ったマーブル模様 | |
| 21-B. Marbling with solver / ソルバーを使ったマーブル模様 | |
| Conclusion / おわりに | 132 |
| About Author / 著者紹介 | 133 |

How to use this book 本書の使い方

Exercise structure 課題の構成

Each exercise is structured with 6 pages in this book. First two pages show the task which you have to solve using Houdini by showing diagram sketches and sample output image. Other 4 pages are used for 2 example solutions for the tasks, which means 2 pages for one solution. A solution page contains a brief description how to solve the task, together with diagram sketches and Houdini network view with a comment.

本書では個々の課題毎に六ページで構成されています。最初の二ページはHoudiniを使って解いてもらいたいタスクの内容が、スケッチやサンプルのアウトプットの画像と共に書かれています。後の四ページは二種類の課題に対する答えのページとして使われています。つまり、一つの課題につき二ページ使っています。課題のページではどのように課題を解くかという簡単な説明と、それに対応したスケッチ、あとコメント付きのHoudiniネットワークを載せています。

Solve tasks first, see answers later まず課題を解き、後で答えを見る

The main purpose for this book is to let readers realize that there is no correct solution to any design problem and realize there is an unlimited number of possibilities for it. So what I want you to do here is that after reading the task pages to understand the problem, I would like you to pause the reading and start thinking about your own solution how you could solve this particular problem before peaking into the solution pages. Launch Houdini and start building your own solution. By doing this you will have a deep understanding what the design task is asking you for.

Two solutions for each task shown in this book is just a reference for you to see that there are always other possibilities for a task. By checking them after you struggle with the task it will give you a broad view when to use what solution. When checking out the solutions I would recommend reading it together with opening HIP files that you can download from the link on the preface page(page 2).

本書のメインのゴールは、読者にどのようなデザインの課題であっても正しい解決方法ではなく、無限の可能性があるということを気づいてもらうことがあります。本書ではまず課題のページを読んで問題を理解してもらったら、答えのページを見る前に一旦本書を読むのを中断し、どのようにその課題を解けるかを考えてもらいたいと思っています。Houdiniを実際に立ち上げ、自分なりに考えた解き方でまず解いてみてください。そうすることで、課題に対する深い理解を得ることができます。

一つの課題につきある二つの答えは、あくまでもその課題を解く上での可能性をしめすための参考として存在しているものです。自分で課題を解いたあとでこれらの参考としての答えを見ることで、どのような場合のときにどのような解法が使えるかというより広い視野を得ることができます。あと答えのページを見るときは、より理解を深めるためにはじめにページ(ページ2)にあるリンクからダウンロードできるHoudiniのHIPファイルと共に読むことをおすすめします。

01. Look at a target

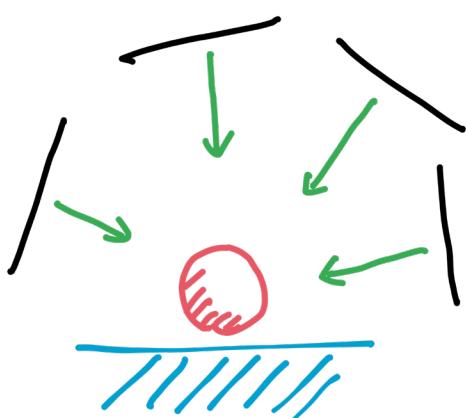
ターゲットに向く

Spread boxes on an XZ plane and rotate each of the boxes to look at the target sphere which could move freely on a 3d space. If possible try to limit the rotational axis to Y-axis (sight angle of the box to be horizontal and avoid looking up or down) even if the target sphere is above or below the box.

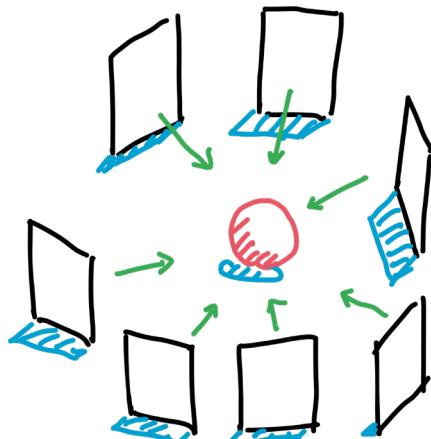
XZ平面上に散らばらせたボックスが、ターゲットである3D空間上自由に動き回れる球体を見る能够るようにそれぞれのボックスを回転させてください。可能であれば、球体がボックスよりも上部あるいは下部にいても、ボックスの回転軸をY軸に限定して目線が上や下を向かないように水平になるようにしてください。

Preferred parameters / 推奨パラメータ:

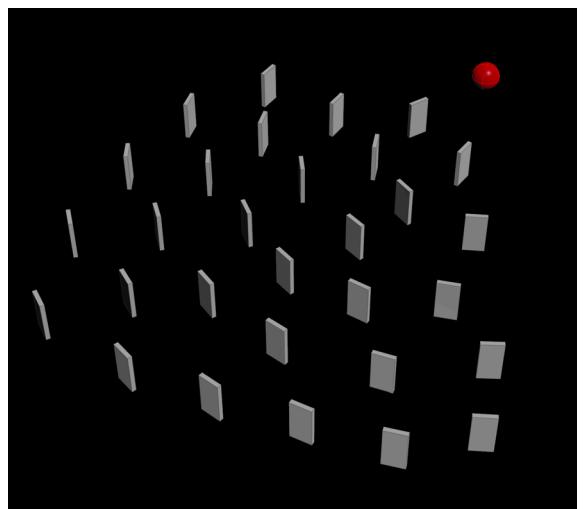
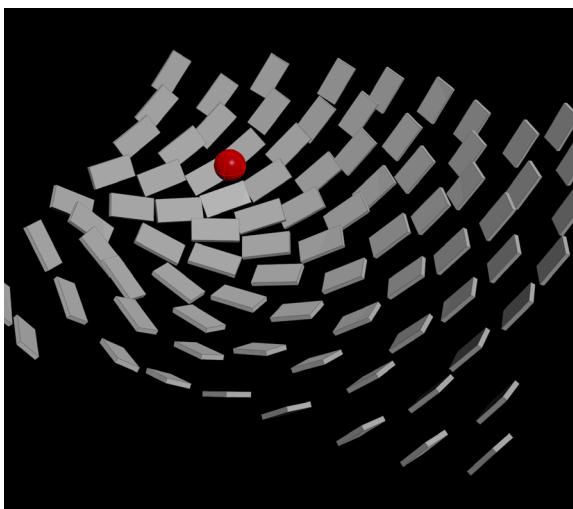
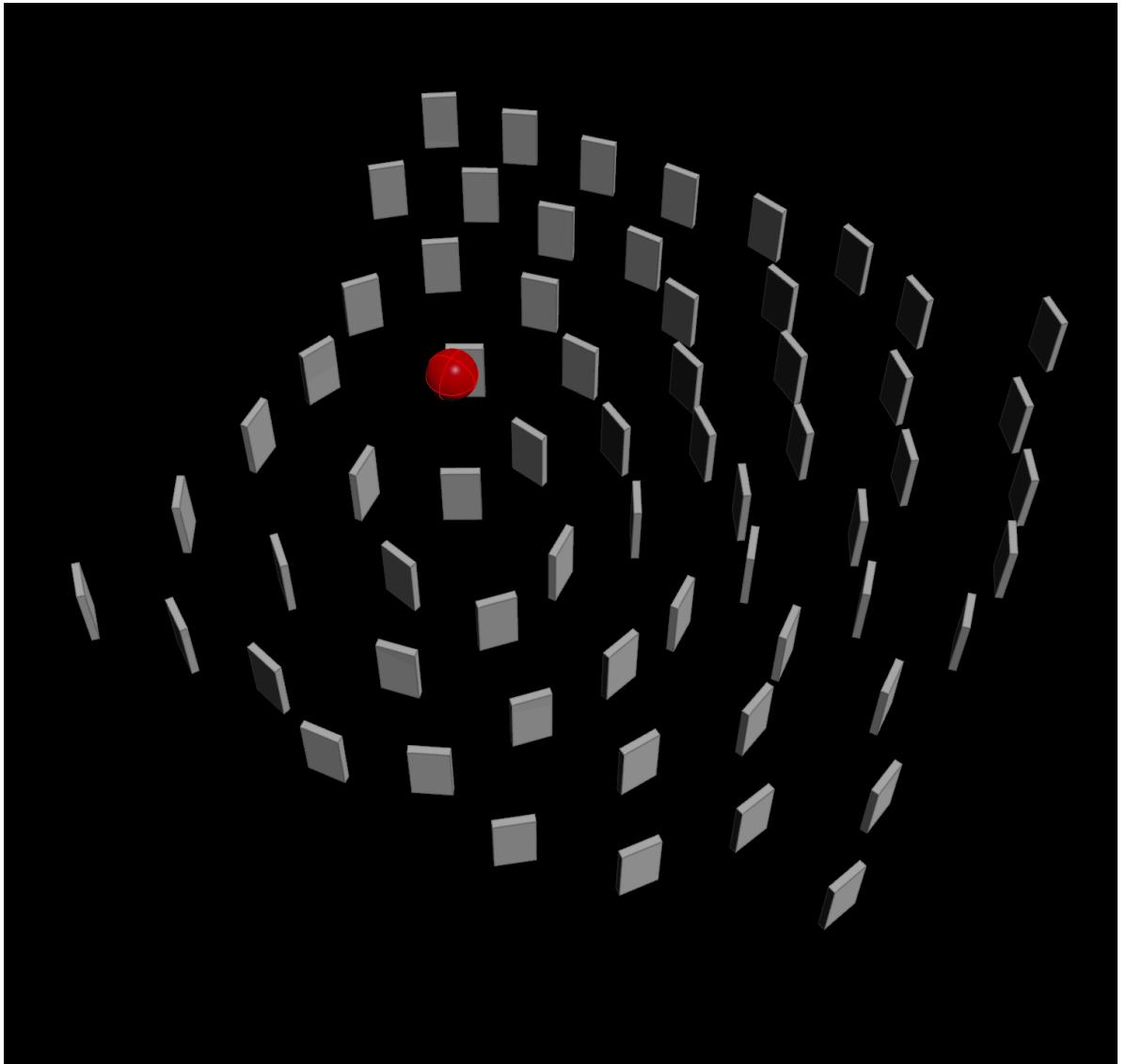
- Number of boxes / ボックスの数
- Target position / ターゲットの位置



Look at a target directly.
ターゲットを直接見る。



Look at a target horizontally.
水平にターゲットを見る。



01-A. Look directly

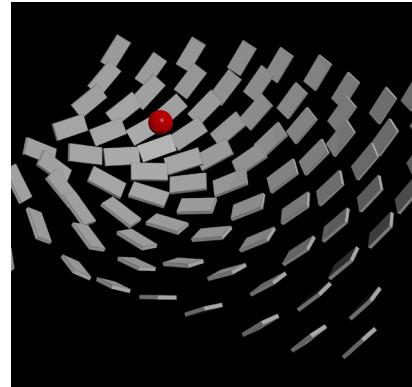
直接見る



01-look_at-A.hiplc

In order to look at a target it is essential to get the look at direction as a vector information. You can easily get that information by subtracting the target position vector by a object (which in this case a box) position vector.

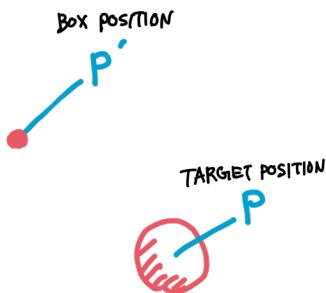
After you've got the look at direction all you have to do next is to set a normal vector N to a point where you want to place a box and use Copy to Point SOP to place box so that the box will rotate itself automatically based on the point's normal direction.



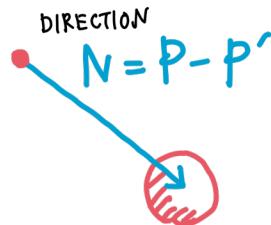
ターゲットに向かって見るためには、基本的には見る方向のベクトルを作つて上げる必要があります。作り方は簡単で、ターゲットの位置ベクトルから見る側の位置(今回の場合はボックスの位置)ベクトルを引き算することで得ることができます。

見る方向のベクトルが得られたら、あとは見る側の位置にあるポイントに法線ベクトルNを設定して、Copy to Point SOPノードでボックスをその位置に移動すれば、自動でボックスがターゲットを向くように回転してくれます。

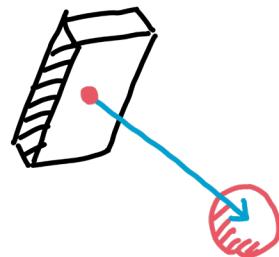
①



②



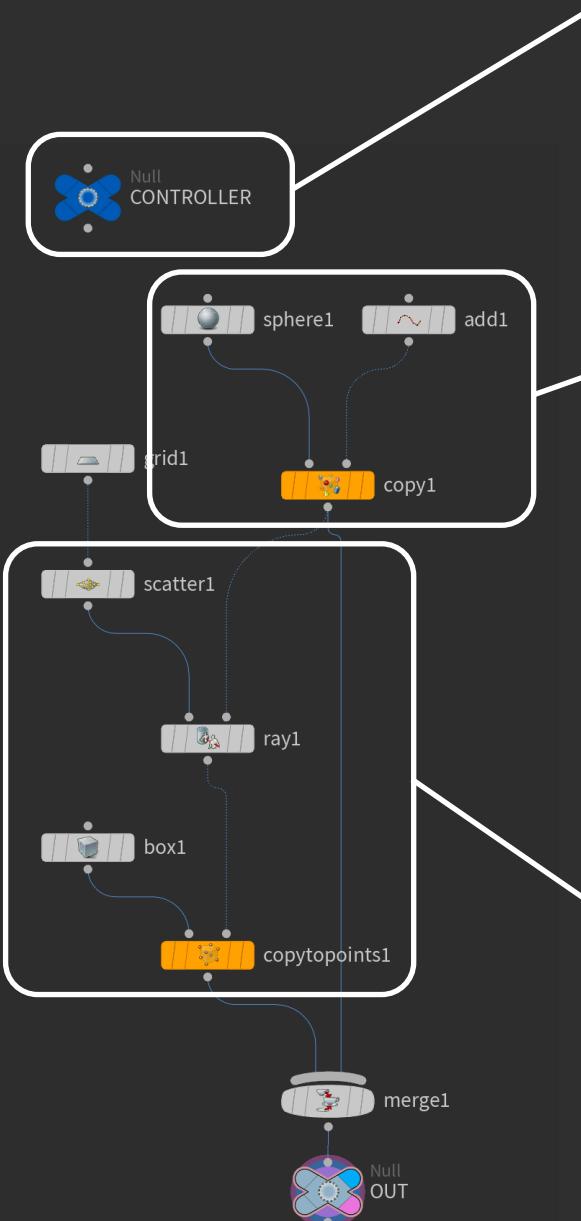
③



Get position of a target and a look from object.
ターゲットと見る元のオブジェクトの位置を取得する

Calculate a look at directional vector.
向く方向のベクトルを計算する。

Use directional vector as a normal vector to rotate box.
向く方向のベクトルをノーマルベクトルとして利用してボックスを回転する。



Setup parameters

- **target_pos**: position of a target
- **pt_num**: number of points to place boxes

パラメータの設定

- **target_pos**: ターゲットの位置
- **pt_num**: ボックスを配置する点の数

Placing target

Create a sphere as a target on a specified position.

ターゲットの配置

指定の位置にターゲットとなる球体を配置します。

Use Ray SOP to create normal

You can create a look at vector by using Ray SOP and set it as normal vector attribute to each scattered point. Then you can use Copy to Point SOP to place box to this point to face the target.

Ray SOPを使って法線を作る

Ray SOPノードを使ってターゲットに向かう方向を法線ベクトルを見る側のポイントのアトリビュートとして作ることができます。その上で、Copy to Point SOPノードを使ってボックスをターゲットに向くように配置します。

01-B. Look horizontally

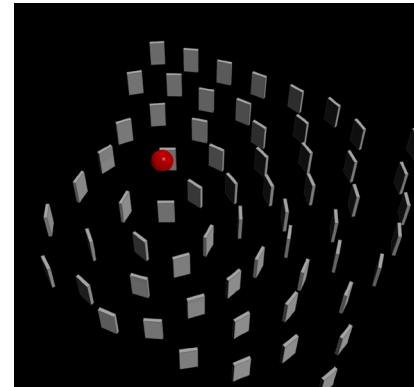
水平を見る



01-look_at-B.hiplc

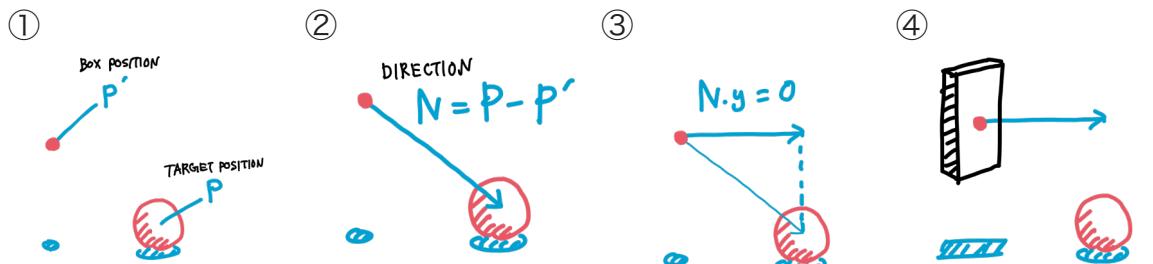
There might be a case you want to avoid boxes to look up or down but only rotating horizontally. One usage could be replacing these boxes as flat texture prop (tree, people ...etc) that should not look up or down.

Easiest way to achieve this is to set the y value of normal vector to 0 so you can force normal vector to always face horizontally. This kind of manipulation can be easily done using VEX.



場合によっては、ターゲットを向くにしても上下には回転させたくない場合もあるかと思います。例えば平面に木や人のテクスチャを貼り付けたようなものが考えられます。

その場合は、最も簡単な方法として考えられるのは、方向ベクトルのY軸の値を0にすることです。そうすることで、ベクトルは常に水平方向(XZ平面)を向くように強制することができます。このようなベクトル操作はVEXを使って簡単に行うことができます。

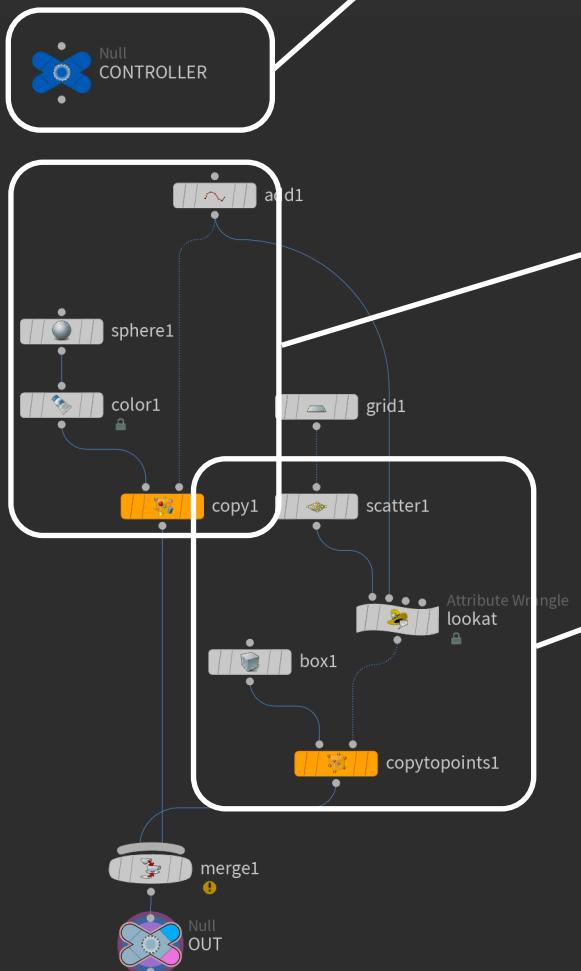


Get position of a target and a look from object.
ターゲットと見る元のオブジェクトの位置を取得する

Calculate a look at directional vector.
向く方向のベクトルを計算する。

Set directional vector's y value to 0.
向く方向のベクトルのYの値を0に設定する。

Use directional vector as a normal vector to rotate box horizontally.
向く方向のベクトルをノーマルベクトルとして利用してボックスを水平に回転する。



Setup parameters

- **target_pos**: position of a target
- **pt_num**: number of points to place boxes

パラメータの設定

- **target_pos**: ターゲットの位置
- **pt_num**: ボックスを配置する点の数

Placing target

Create a sphere as a target on a specified position.

ターゲットの配置

指定の位置にターゲットとなる球体を配置します。

Create normal vector using VEX

Calculating normal vector to look at a target horizontally can be easily done by using VEX via Point wrangle node with simple equations.

VEXを使って法線を作る

ターゲットに向かって水平に向くような法線ベクトルはPoint wrangleノードでVEXを書くことで比較的簡単に実現することができます。

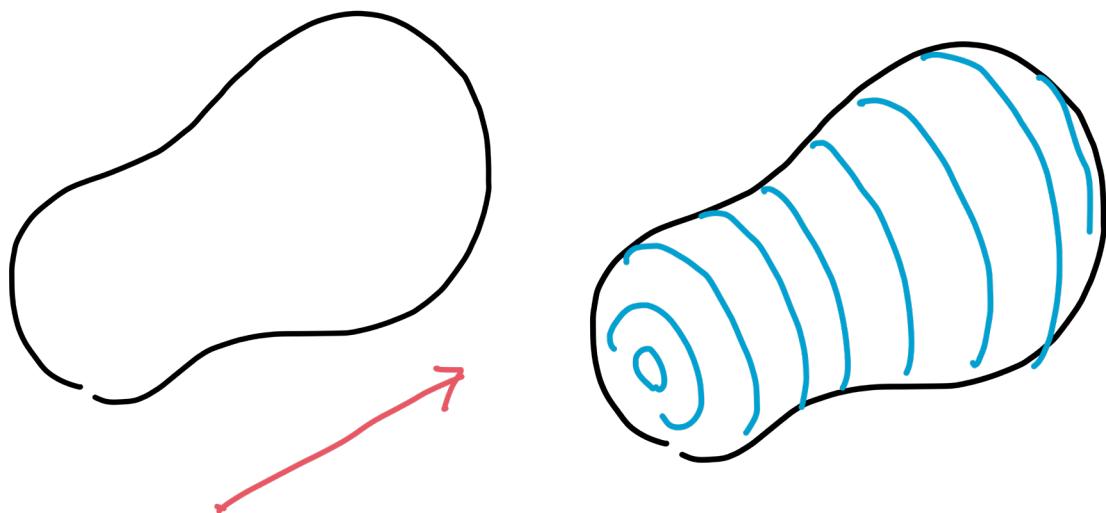
02. Contouring オブジェクトをコンター割りする

Create a contouring (sliced) surfaces from an any input geometry. You should be able to change the number of contours (or contouring step distance) and if possible a direction to create a contours.

好きなジオメトリからコンター(輪切り)サーフェスを作ってください。その際、コンターの数(あるいはコンター間の大きさ)と、可能であればコンターを作る方向を指定できるようにしてください。

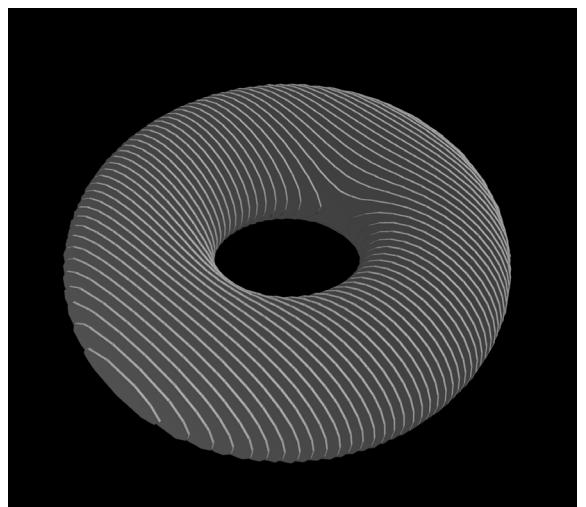
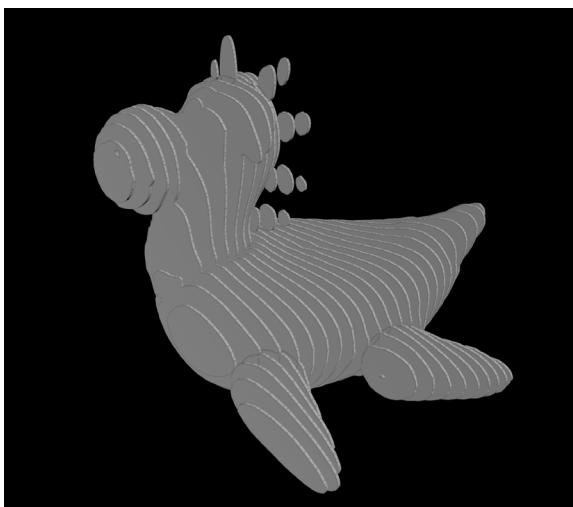
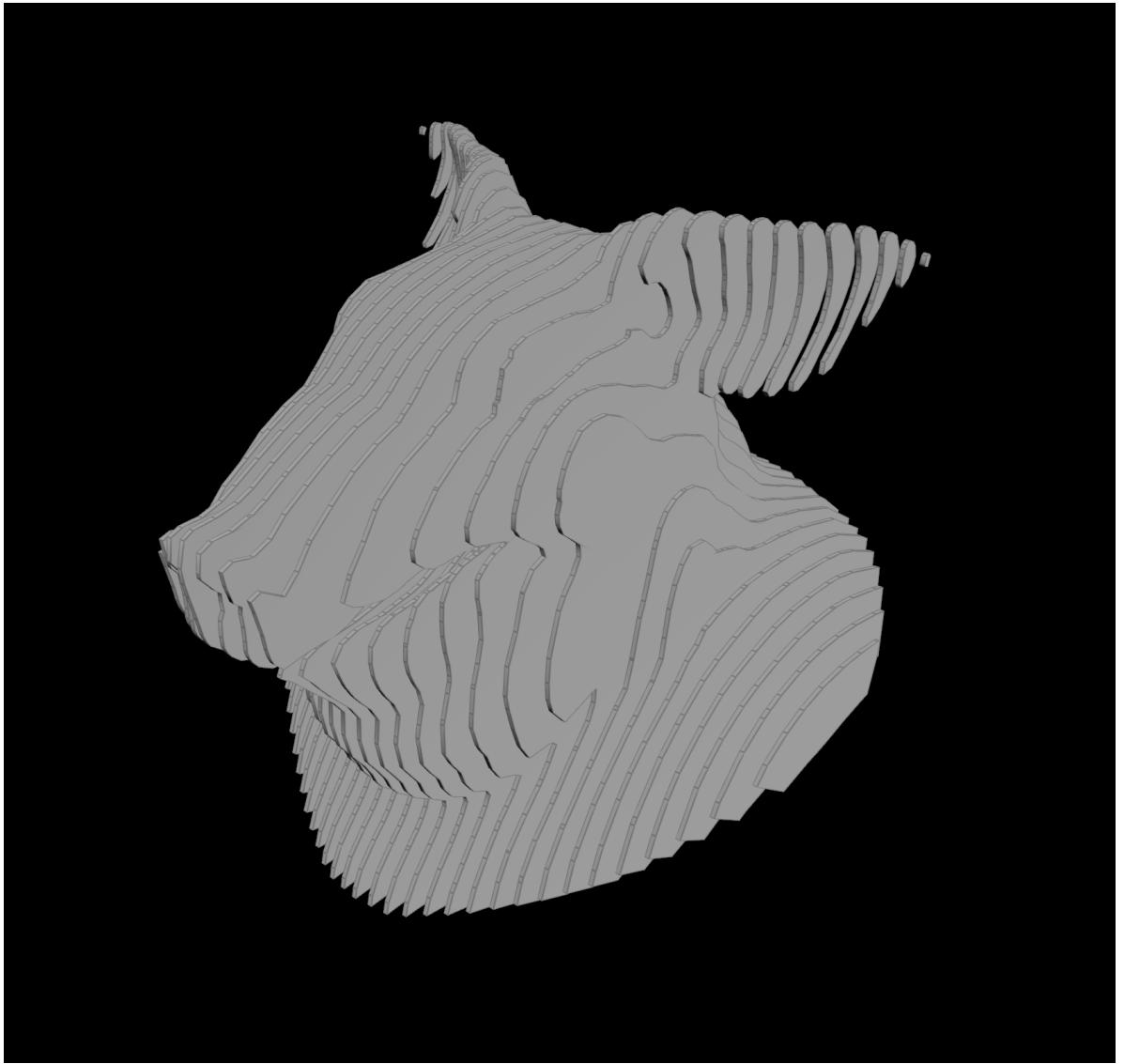
Preferred parameters / 推奨パラメータ:

- Number of contours / コンターの数
- Direction to create contours / コンターを作る方向



Decide the direction to contour.
コンタリングしたい方向を決定する。

Make contourings.
コンターを作る。



02-A. Contouring with loop

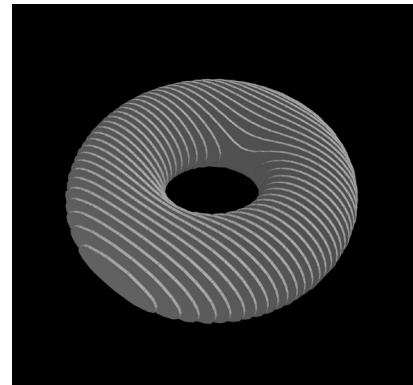
ループを使ってコンター割りする



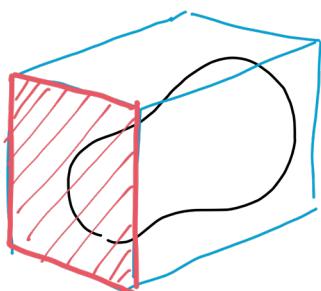
02-contouring-A.hiplc

One of a way to create a contour is to use a loop and in Houdini you can use For Each loop for looping. One example I could think of is to extract one of a face from a bound of an input geometry and use it to extract sections of the geometry inside For Each loop with Transform SOP and Boolean SOP.

コンターを作るひとつ的方法として、ループを使うのが考えられます。HoudiniではFor Eachループを使います。その例のひとつとして、任意のジオメトリのバウンディングボックスの一つの面を取り出して、それをFor Eachループの中でTransform SOPで動かしBoolean SOPでジオメトリの断面を取り出すという方法が考えられます。

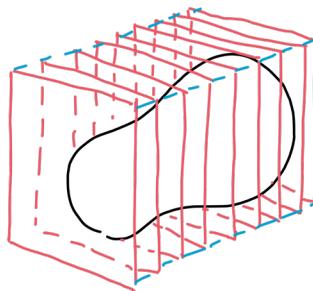


①



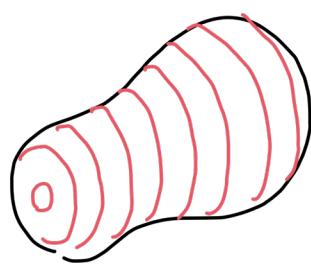
Get a face from boundary.
バウンディングボックスから面を取得する。

②

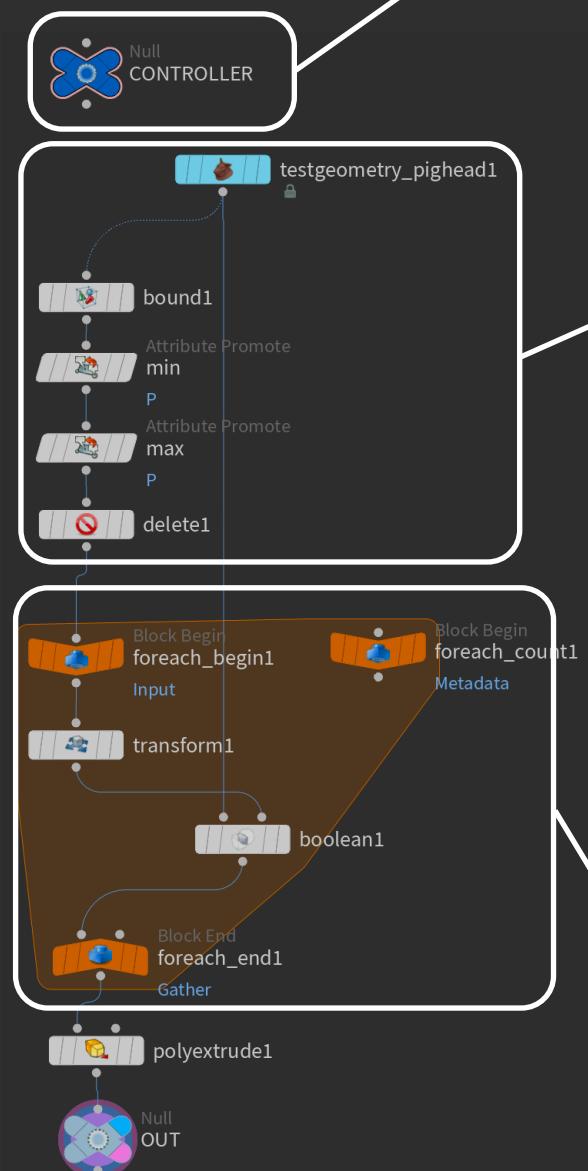


Copy face to fill bounding
box as layers.
面をバウンディングボックスに層として配置する。

③



Use Boolean SOP to get
intersection between faces
and geometry.
ジオメトリとコピーした面の交差をBoolean SOPで得る。



Setup parameters

- **contour_num**: number of contours
- **thickness**: thickness of contour surface

パラメータの設定

- **contour_num**: コンターの数
- **thickness**: コンターフェースの厚さ

Extract a face from a bound

Create a bounding box from an input geometry and retrieve one of a face. Also get min and max position of a bound points to use it to calculate the distance how much it should move.

バウンディングボックスから面を取り出す
任意のジオメトリからバウンディングボックスを作り、その面の一つを取り出します。同時に、面の移動距離を計算するためのバウンディングボックスの点の最小最大位置情報を取得します。

Use loop to cut section

Use For Each loop to move the extracted face using calculated min and max position and use Boolean SOP to retrieve a section of the input geometry.

ループを使って断面を切る

For Eachループを使って、先程取り出した面をバウンディングボックスの点の最小最大位置情報を用いて計算して移動し Boolean SOPでジオメトリの断面を切り出します。

02-B. Contouring simultaneously

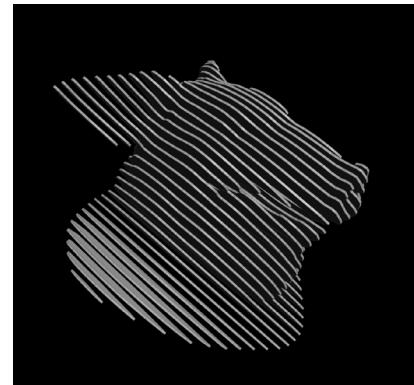
一気にコンター割りする



02-contouring-B.hiplc

Another way to create contours is to use Copy SOP to simultaneously create sections. In this way it calculates a bit faster than using loop. In this example I've also added additional parameter so that you can specify the direction of the contouring.

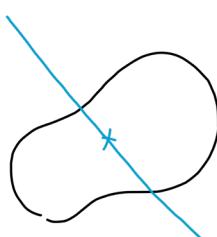
In particular use Line, Bound and Intersection Analysis SOP to create a core line and place number of planes vertically on the line to use it to cut the section from an input geometry.



コンターを作る他の方法として考えられるのは、Copy SOPを使って一気に作るという方法です。この方法を使うとFor Eachループを使うよりも多少速く計算することができます。この例では、パラメータとしてコンターの向きも加えています。

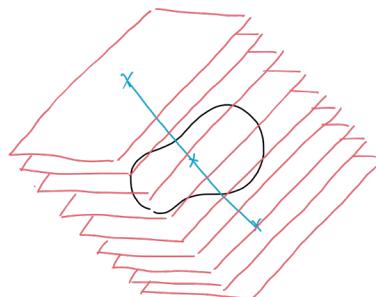
具体的にはLineとBoundとIntersection Analysis SOPを使ってコンターを作るための芯となる線を作り、そこに垂直に複数の平面を配置します。その平面をつかって任意のジオメトリの断面を切り取ります。

①



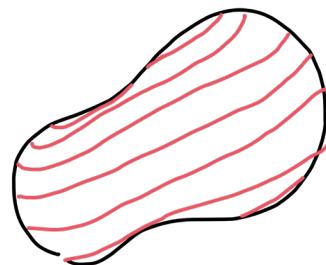
Create a line determines a direction for contours on the center of geometry.
コンターの方向を決めるラインをジオメトリの中心に配置する。

②

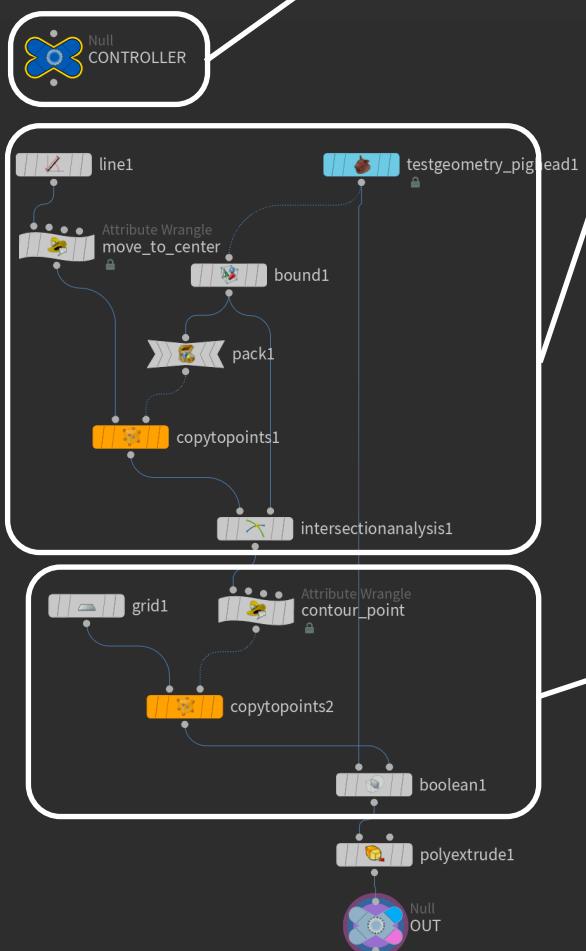


Place layer of planes on the line.
ラインに複数層の平面を配置する。

③



Create contours from the geometry and the planes.
ジオメトリと複数の平面からコンターを作る。



Setup parameters

- **contour_dir:** contour direction
- **contour_num:** number of contours
- **thickness:** thickness of contour surface

パラメータの設定

- **contour_dir:** コンターの方向
- **contour_num:** コンターの数
- **thickness:** コンタ一面の厚さ

Create core line for the contour

Based on the contour direction parameter create a core line on the center of input geometry.

コンターのための芯の線を作る

コンターの方向のパラメータに応じて芯の線を任意のジオメトリの中心に作ります。

Create contours simultaneously

Use Copy SOP to place number of plane surfaces vertically on the core line and use Boolean SOP to create sections as a contour surface.

一気にコンターを作る

Copy SOPを使って複数の平面を垂直に芯の線に配置します。その平面をBoolean SOPに介して使って任意のジオメトリの断面を切り取ります。

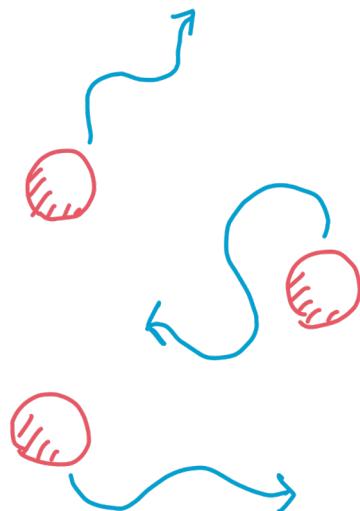
03. Random Walk ランダムに歩く

Create multiple objects to randomly walk (randomly choose direction per frame to move) inside a specified box region with specified speed and . If possible let the object comes out from the other side of the box region when the object goes outside the box.

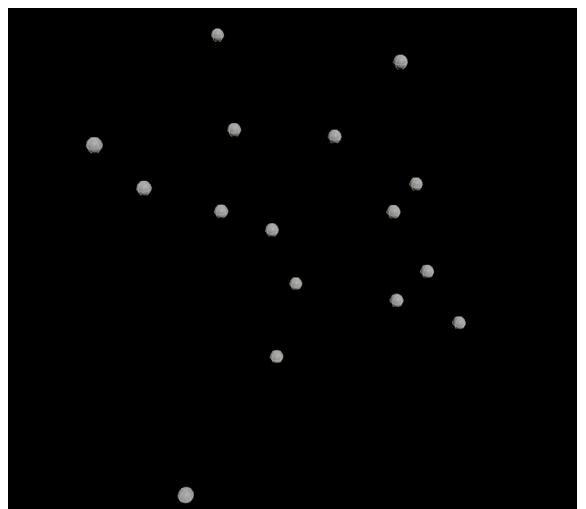
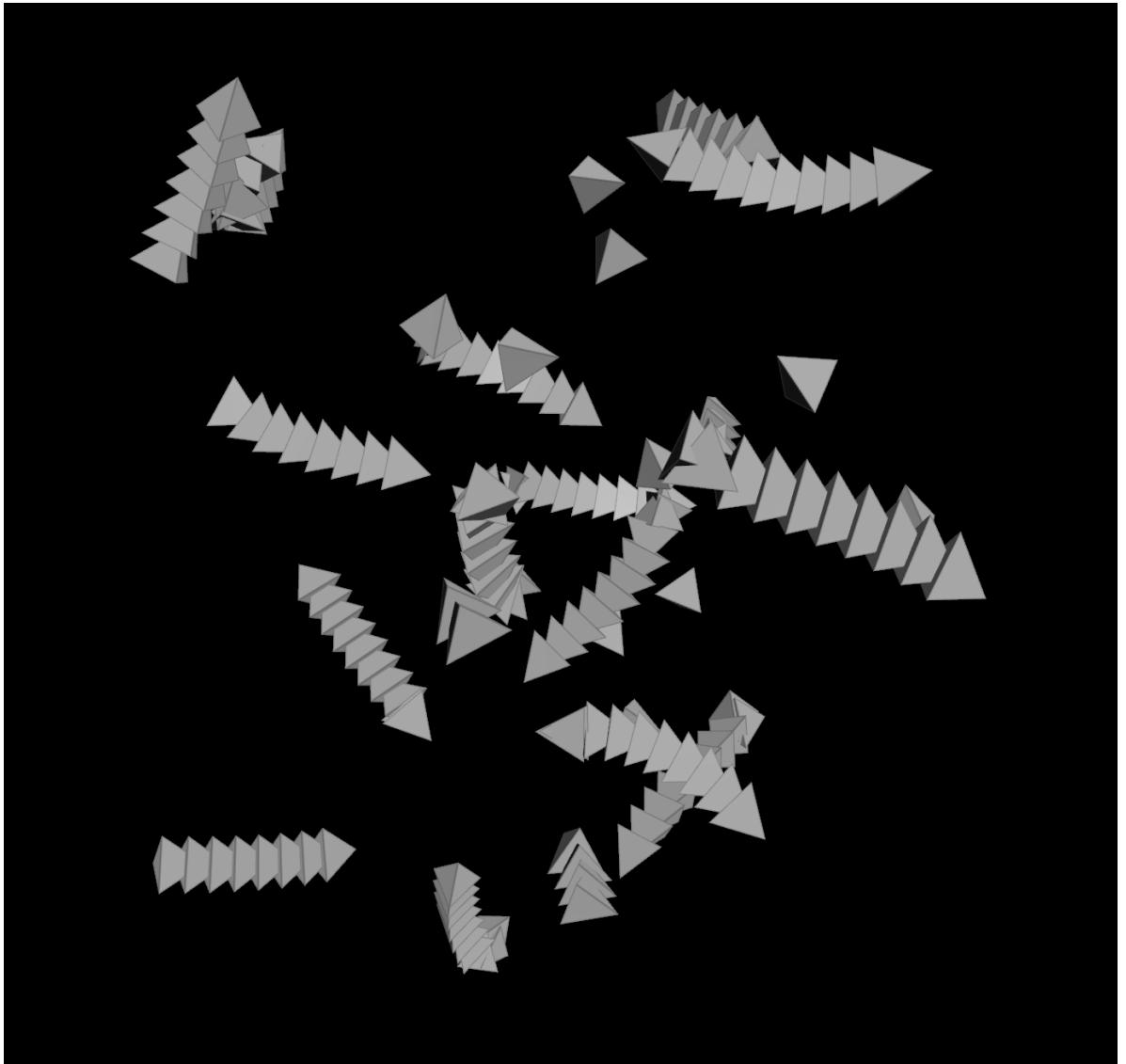
指定のボックスの領域の中をランダムに歩く複数の任意のオブジェクト(例えば球体)を作ってください。その時、スピードや歩く方向のランダム具合をコントロールできるようにしてください。もしオブジェクトがボックス領域の外に出たら、反対側の位置からオブジェクトが出てくるようにしてください。

Preferred parameters / 推奨パラメータ:

- Number of objects to walk randomly / ランダムに歩くオブジェクトの数
- Walking speed / 歩くスピード
- Walking direction randomness / 歩く方向のランダム具合



Let objects walk randomly.
ランダムにオブジェクトを歩かせる。



03-A. Random walk with solver

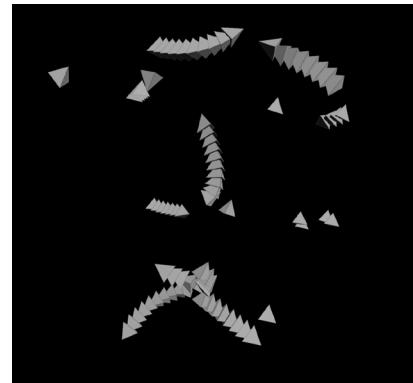
ソルバーを使ってランダムに歩く



03-random_walk-A.hiplc

One of a way to let object randomly walk in ease is to use Solver SOP together with some noise function to determine the moving direction for each frame.

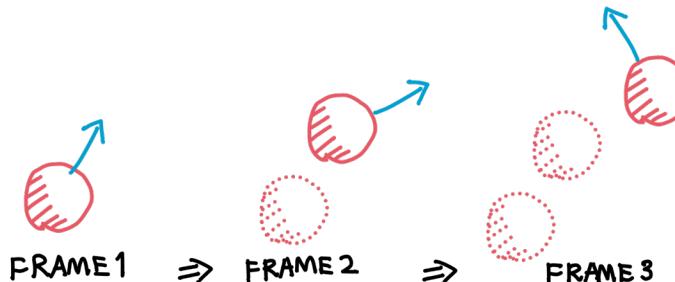
A point here is to write a process to move randomly inside a solver network which will be recursively called each frame and I'm using VEX to write a simple code to let point randomly moves using noise function because of it's flexibility but this process could also be replaced using Attribute Noise SOP.



オブジェクトを簡単にランダムに歩かせる方法の一つとして考えられるのはSolver SOPを使って、ある種のノイズ関数を利用して歩く方向を毎フレームランダムに決定することです。

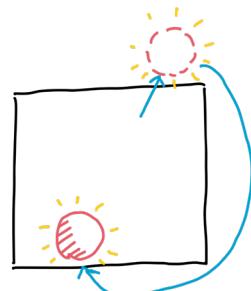
ここで重要なのは、ソルバーネットワークの中でポイントをランダムな方向に動かすプロセスを書けば、毎フレーム再帰的にその同じプロセスが呼ばれるという点です。この例ではその自由度の高さからVEXを使ってランダムに動くシンプルなコードを書いていますが、同じことをAttribute Noise SOPノードを使って書き換えることも可能です。

①

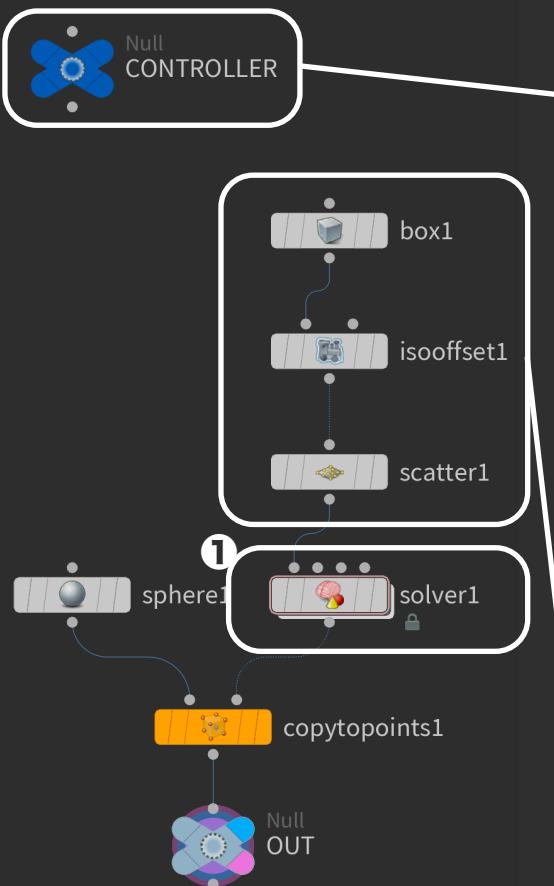


Move object to random direction per frame.
毎フレーム、オブジェクトをランダムな方向に移動する。

②



Move object to opposite side of boundary when it goes out the boundary.
オブジェクトが領域の外に出たら
領域の反対の位置に移動する。



Setup parameters

- **size:** size of a boundary for points to walk
- **pt_num:** number of points to walk
- **speed:** walking speed
- **randomness:** how random points walk

パラメータの設定

- **size:** ポイントが歩ける領域のサイズ
- **pt_num:** ポイントの数
- **speed:** 歩くスピード
- **randomness:** 歩く方向のランダム具合

Create points to walk

Create number of points that you want to let walk inside a boundary.

歩かせる点を作る

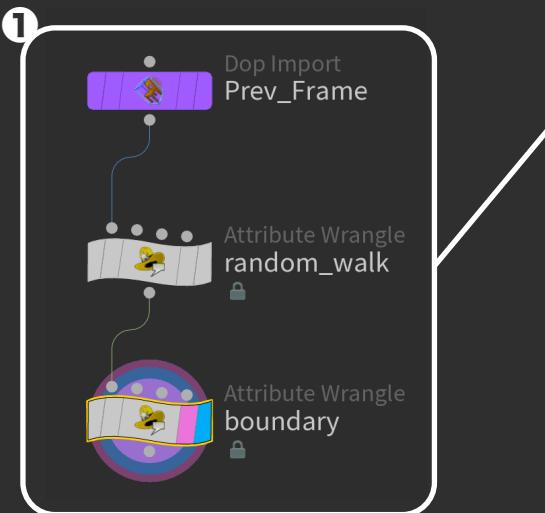
歩かせたい複数の点を歩ける領域の中に作ります。

Random walk using VEX

Write a simple code to randomly move points using noise function in VEX based on point's position, frame count and point's number. Also add another Point wrangle to keep points inside the boundary by letting points jump to the other side of the boundary when it goes out.

VEXを使ってランダムに歩かせる

VEXのノイズ関数を利用してポイントの位置やフレーム数、ポイントの番号を利用してポイントをランダムに移動します。追加で Point Wrangleノードを利用して、ポイントが設定した領域の外に出たときに領域の反対側に移動するようにします。



03-B. Random walk with noise

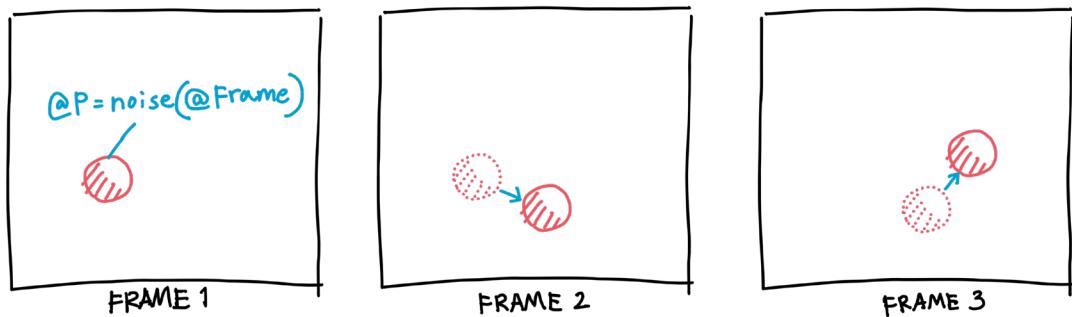
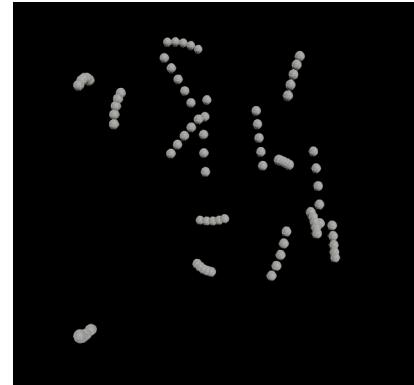
ノイズ関数を使ってランダムに歩く



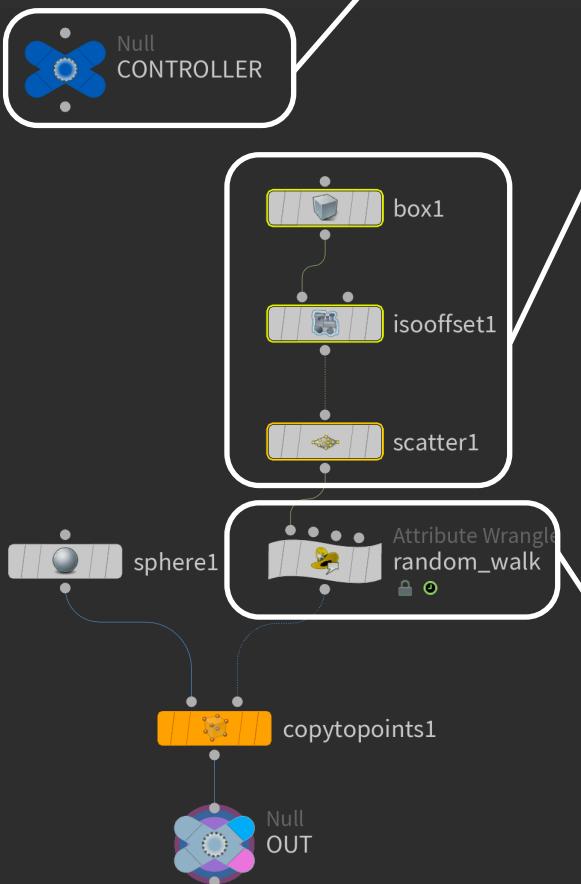
03-random_walk-B.hiplc

Another way to let points randomly walk is to just use VEX with noise function without solver network. Last example used noise function to determined the random walking direction but this time example uses noise function to determine the position itself of points.

ポイントをランダムに歩かせる別の方法としてソルバーネットワークを使わずにVEXのノイズ関数だけを利用する方法が考えられます。前の例ではランダムに歩く際の方向をノイズ関数を使って作っていましたが、今回はポイントの位置自体をVEXを使って作ります。



Use noise function with frame value in VEX to determine the position for the object at each frame. VEXのnoise関数をフレームの値と共に利用してオブジェクトの位置を毎フレーム決定します。



Setup parameters

- **size**: size of a boundary for points to walk
- **pt_num**: number of points to walk
- **speed**: walking speed

パラメータの設定

- **size**: ポイントが歩ける領域のサイズ
- **pt_num**: ポイントの数
- **speed**: 歩くスピード

Create points to walk

Create number of points that you want to let walk inside a boundary.

歩かせる点を作る

歩かせたい複数の点を歩ける領域の中に作ります。

Update point position with noise

Use Point Wrangle node to update point position using noise function based on frame count and point number.

ノイズを使ってポイントの位置を更新する

Point Wrangleでノイズ関数を使ってポイントの位置をポイントの番号とフレーム数を利用してアップデートします。

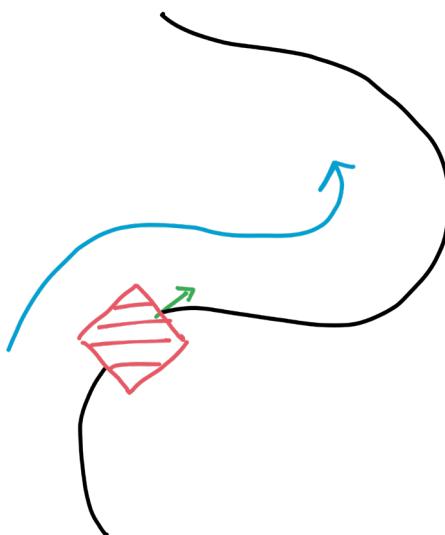
04. Move along a path パスに沿って動かす

Move any object along a curve path you provided from starting point to end point of the curve. When moving the object try to rotate the object based on the tangent direction of the path.

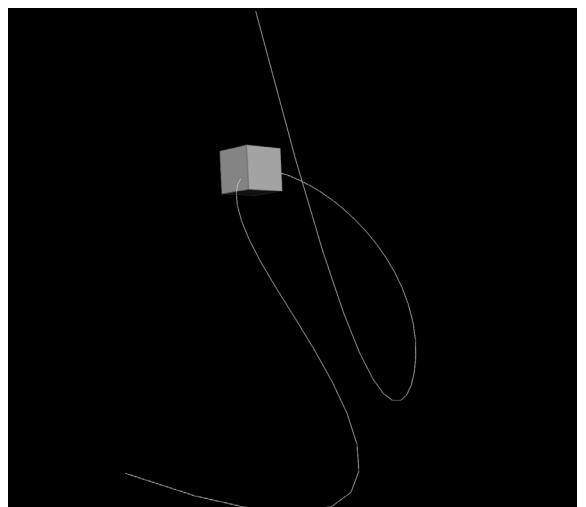
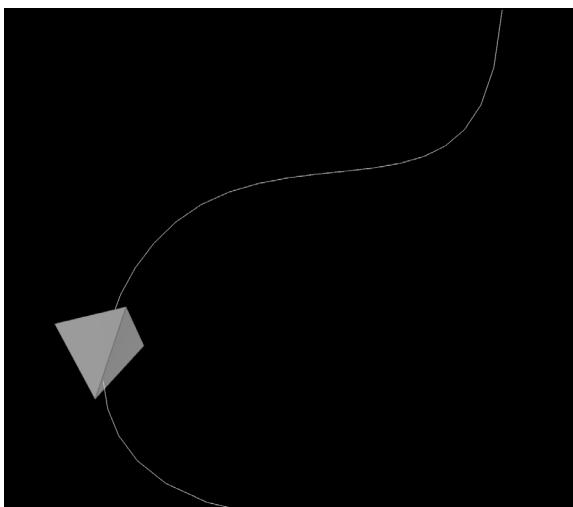
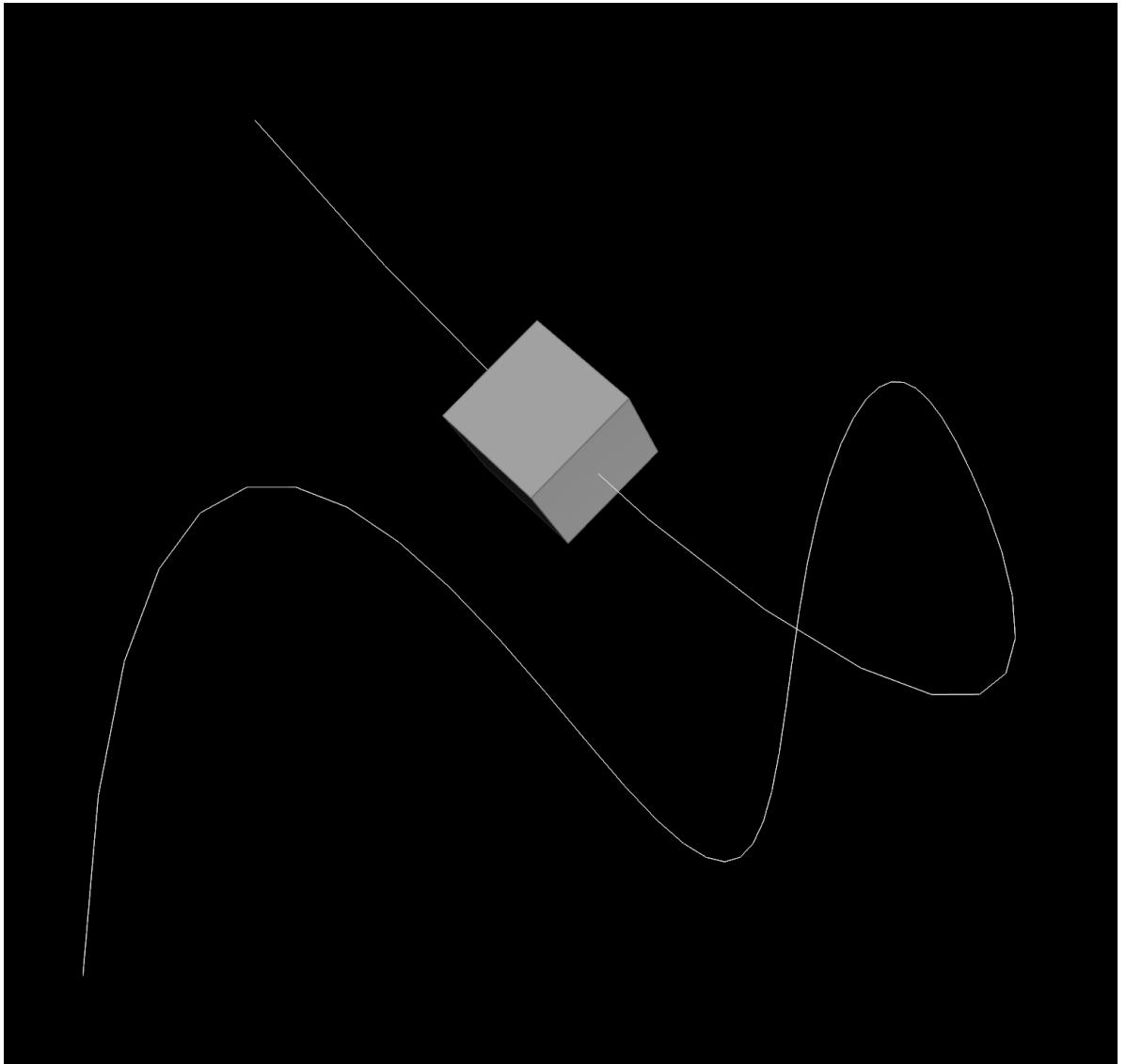
曲線パスに沿って、始点から終点にかけてオブジェクトを動かせるようにしてください。その時、オブジェクトの向きが曲線の接線方向に沿うようにしてください。

Preferred parameters / 推奨パラメータ:

- Position on a curve path to place object / オブジェクトを配置する曲線上の位置



Move object along a path.
オブジェクトをパスに沿って動かす。



04-A. Move object with Carve

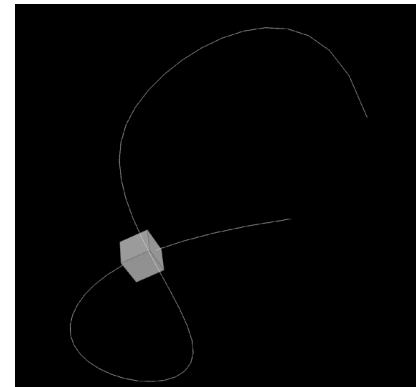
Carveを利用してものを動かす



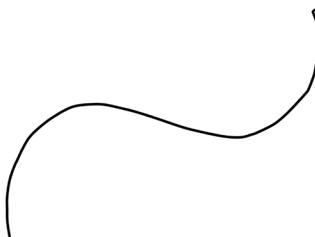
04-path_move-A.hiplc

In this example I'm using Carve SOP to determine the specific position on a curve where you want to move the object to. This is one of a simplest way to move object along the curve.

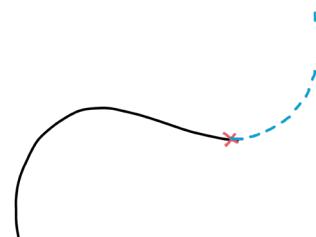
この例ではCarve SOPを使ってカーブ上の点を取得し、その点にオブジェクトを移動します。この方法が考えられる中でも最もシンプルな方法のひとつです。



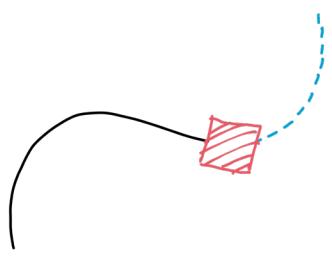
①



②



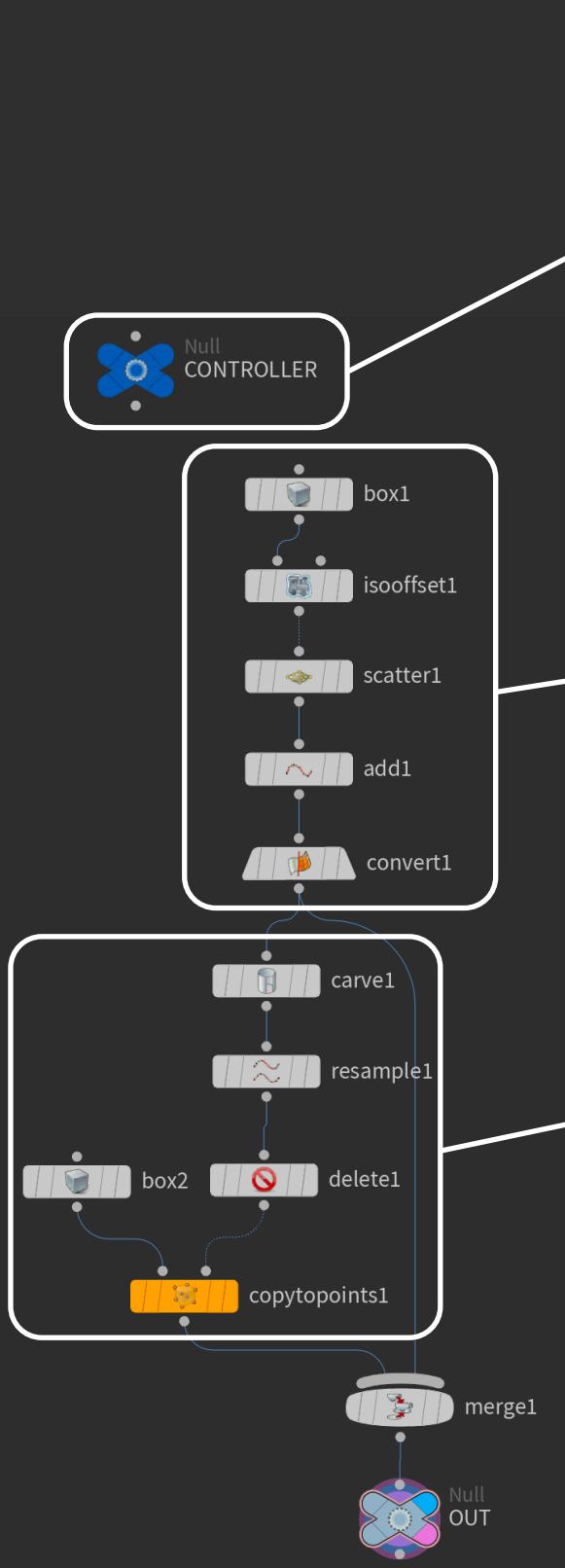
③



Create a free curve.
自由曲線を作る。

Use Carve SOP to trim curve
to get new end point.
Carve SOPを使って、曲線をトリムして新しい端点を作る。

Place object on a new end
point.
新しい端点にオブジェクトを配置する。



Setup parameters

- **pt_num:** number of curve points
- **seed:** random seed for points position
- **path_pos:** position on curve to move object to

パラメータの設定

- **pt_num:** カーブの頂点の数
- **seed:** カーブ頂点の位置のランダムシード
- **path_pos:** オブジェクトを移動するカーブ上のポイント

Create random curve path

Create random curve path from a number of points randomly placed inside a box.

ランダムな曲線パスを作る

ボックスの中にランダムに配置されたポイントを複数配置して、それを結ぶ曲線を作ります。

Move object to a point on curve

Use Carve SOP to get a point on a curve and move object to this point.

カーブ上の点にオブジェクトを移動する

Carve SOPを利用してカーブ上の点を取得し、その点にオブジェクトを移動します。

04-B. Move object with curve uv

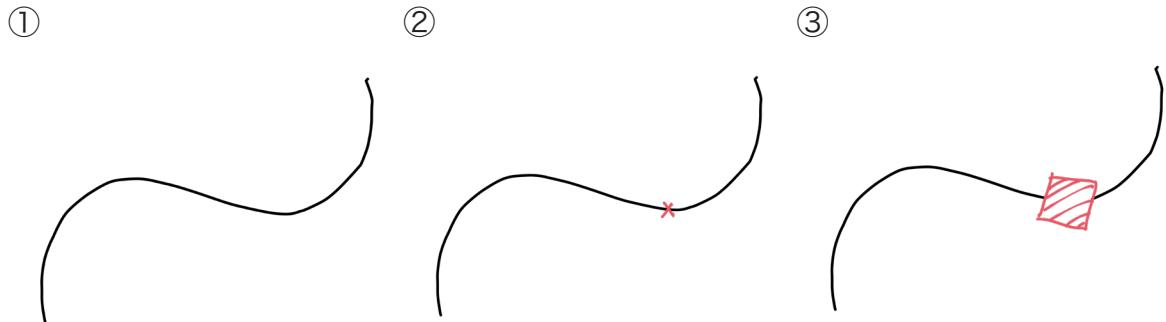
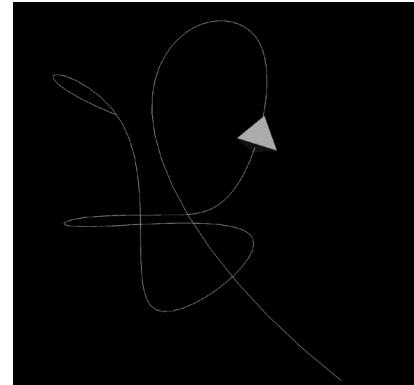


曲線のUVを使ってものを動かす

04-path_move-B.hiplc

In this example I'm using curve uv to retrieve a position on a curve in order to move an object. I'm using VEX to specify a position on a curve by an uv parameter.

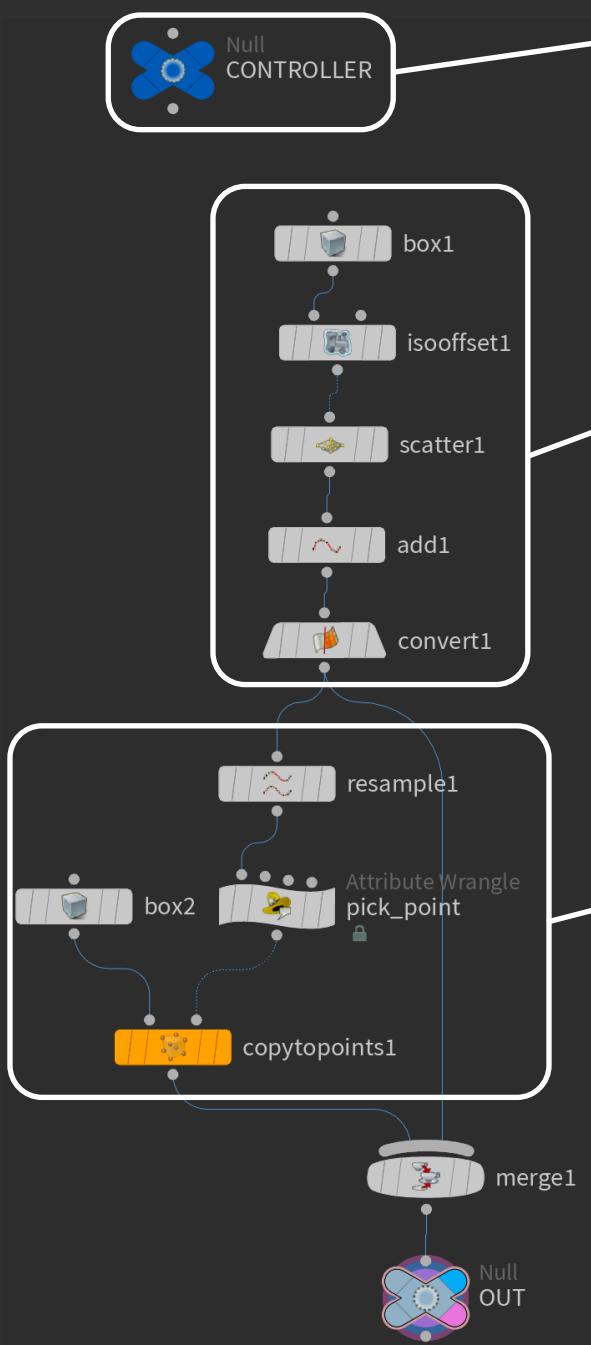
この例では曲線のUVを利用してカーブ上のポイントの位置を取得し、そこにオブジェクトを移動しています。具体的にはVEXを介してuvパラメータからカーブ上のポイントの位置を取得しています。



Create a free curve.
自由曲線を作る。

Create a point on a curve
using uv parameter with
VEX.
VEXとuvのパラメータを利用して
曲線上のポイントを取得する。

Place object on the created
point.
作ったポイントにオブジェクトを
配置する。



Setup parameters

- **pt_num**: number of curve points
- **seed**: random seed for points position
- **path_pos**: position on curve to move object to

パラメータの設定

- **pt_num**: カーブの頂点の数
- **seed**: カーブ頂点の位置のランダムシード
- **path_pos**: オブジェクトを移動するカーブ上のポイント

Create random curve path

Create random curve path from a number of points randomly placed inside a box.

ランダムな曲線パスを作る

ボックスの中にランダムに配置されたポイントを複数配置して、それを結ぶ曲線を作ります。

Move object to a point on curve

Use Wrangle node to get a point on a curve by writing a VEX code with uv parameter and move object to this point.

カーブ上の点にオブジェクトを移動する

WrangleノードでVEXコードを書いてuvパラメータからカーブ上のポイントの位置を取得し、そこにオブジェクトを移動します。

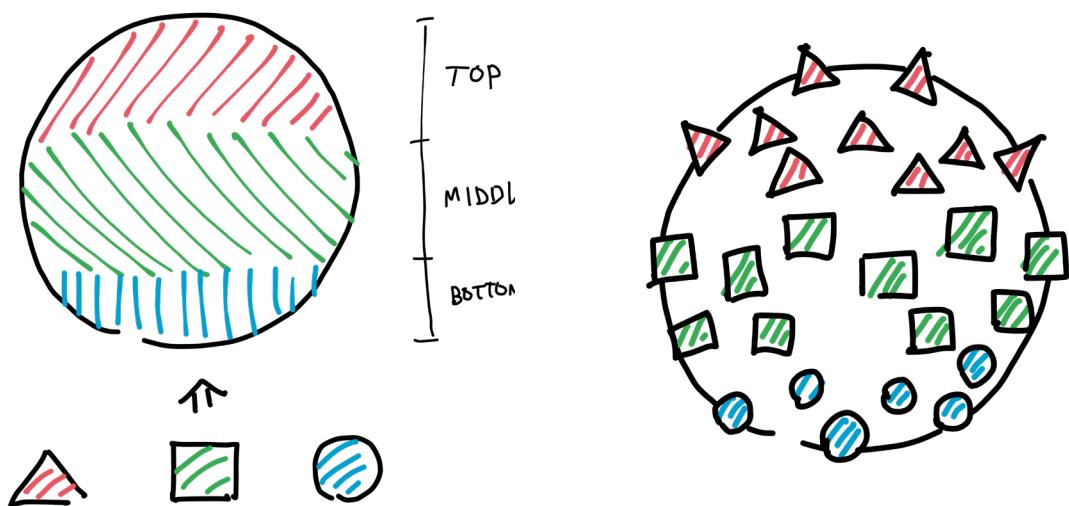
05. Switch objects 条件に応じてオブジェクトをスイッチする

Place different types of object on an input geometry based on a condition. In this case, create number of points on a geometry and place an object chosen from three types (sphere, box and pyramid) based on the verticalness of a normal direction of each point.

任意のジオメトリに種類の違うオブジェクトを条件に応じて配置してください。今回の場合は、まずジオメトリの上に複数のポイントを作り、そのポイントの法線方向の垂直具合に応じて三種類のオブジェクトの中から選んでそのポイントの位置に配置してください。

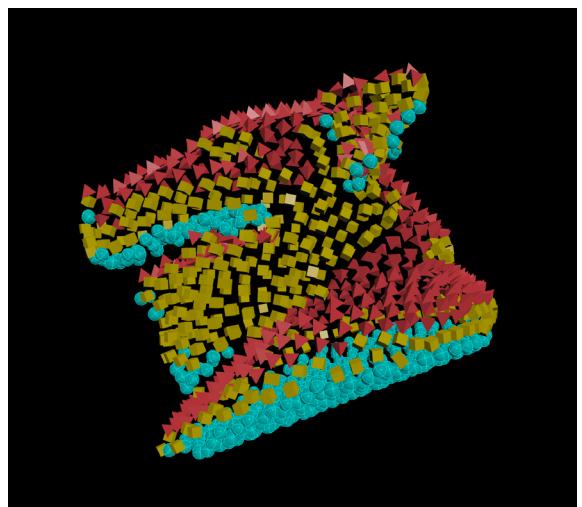
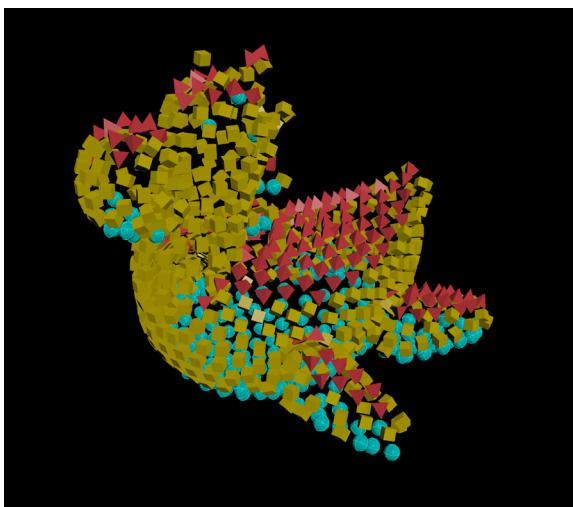
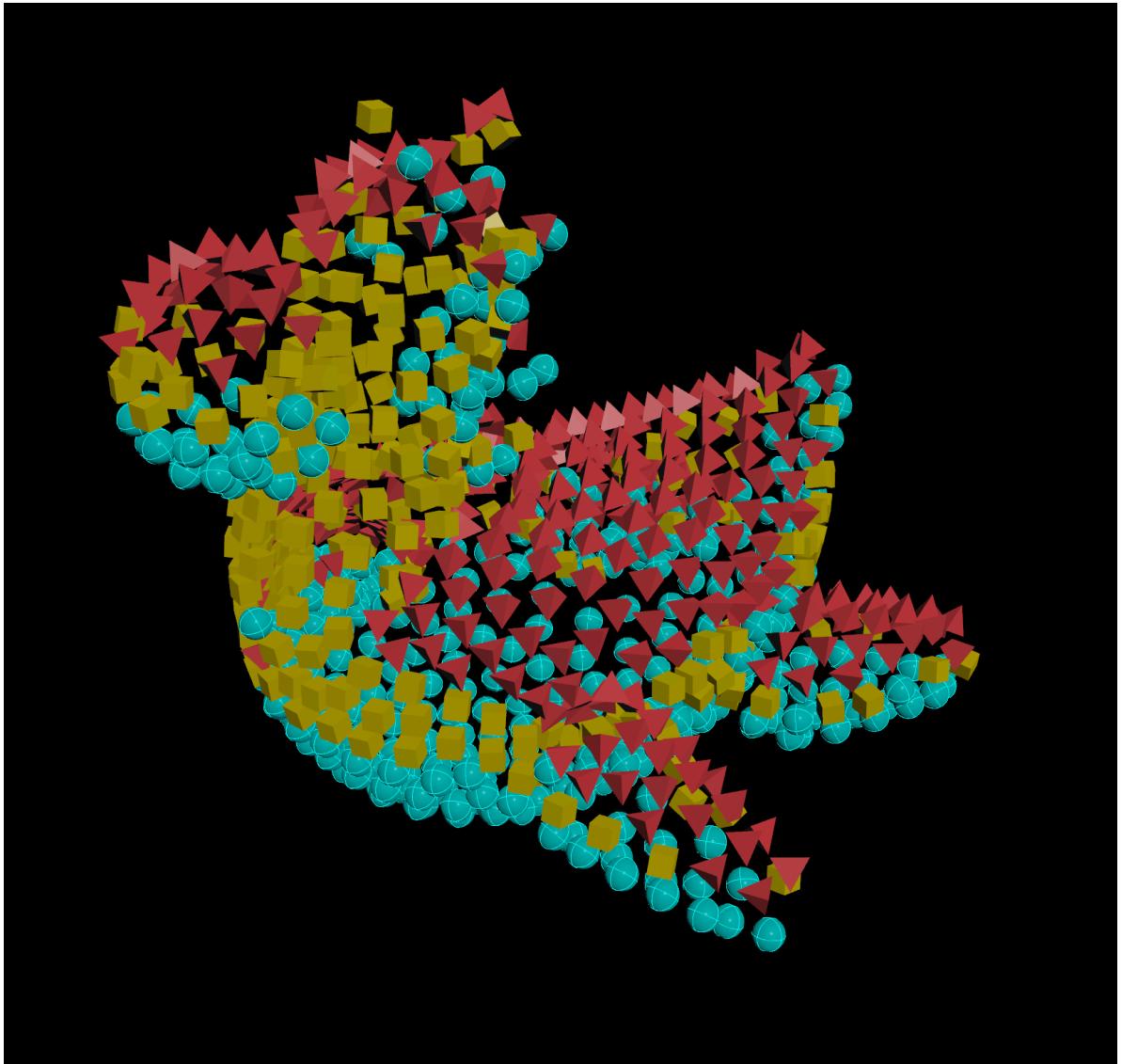
Preferred parameters / 推奨パラメータ:

- Number of object / オブジェクトの数
- Angle (verticalness) thresholds / 角度(垂直具合)パラメータ



Place specific type of object on a geometry based on the verticalness of geometry's point normal direction.

ジオメトリのポイントの法線方向の垂直性を利用し、それに応じたタイプのオブジェクトをジオメトリに配置する。



05-A. Switching with Stamp

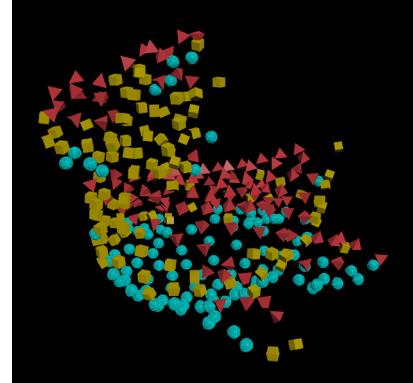
Stampを利用してスイッチする



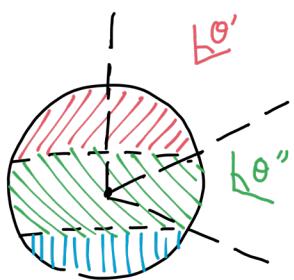
05-switch-A.hiplc

In this example I'm using Switch SOP together with Copy Stamp SOP to place three types of objects on an input geometry based on a condition. In particular the condition here is based on verticalness of a normal direction of points of the geometry which can be determined with a Group SOP with Keep by Normals option.

この例ではSwitch SOPとCopy Stamp SOPを利用して三種類のオブジェクトを任意のジオメトリの上に条件に応じて配置しています。具体的には、ジオメトリ上のポイントの法線ベクトルの垂直性をGroup SOPのKeep by Normalsオプションを利用することで、どのポイントにどのオブジェクトを配置するかを決定します。



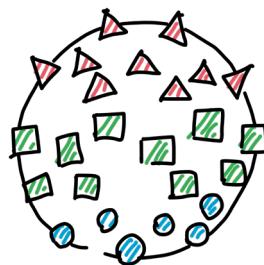
①



GROUP SOP
+ Keep by Normals

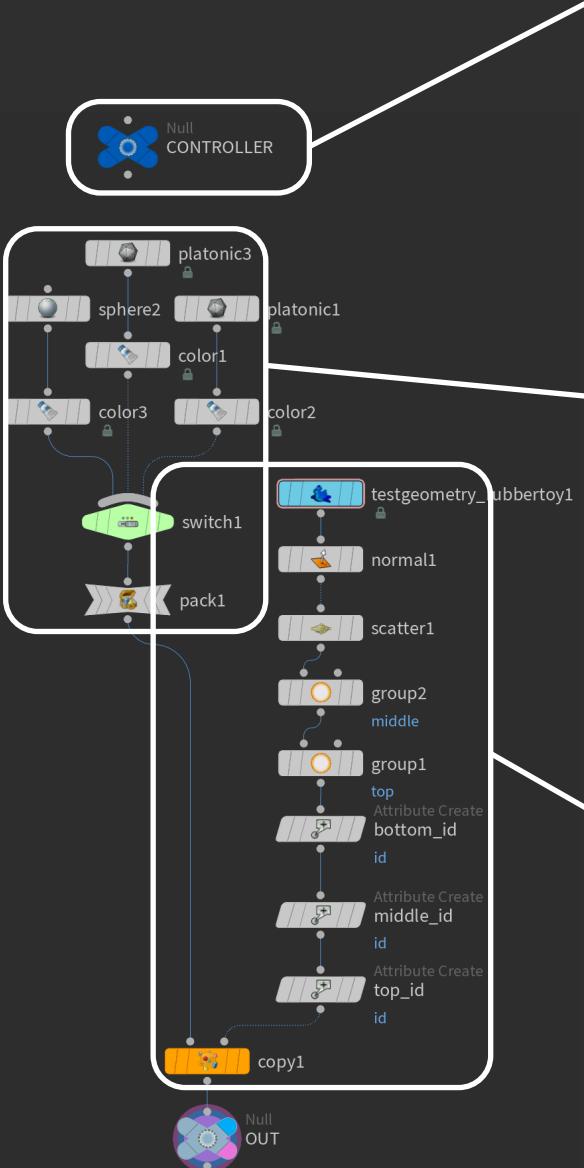
Use Group SOP with Keep by Normals option to create three types of group to points on a geometry.
Group SOPのKeep by Normalsオプションを利用してジオメトリ上のポイントに三つのタイプのグループを作る。

②



COPY STAMP

Place three types of object on a geometry using Copy Stamp SOP.
Copy Stamp SOPを利用して三つのタイプのオブジェクトをジオメトリの上に配置する。



Setup parameters

- **pt_num:** number of points on geometry
- **top_angle:** vertical angle condition for first object type to be placed
- **middle angle:** vertical angle condition for second object type to be placed

パラメータの設定

- **pt_num:** ジオメトリ上のポイントの数
- **top_angle:** 1つ目のタイプのオブジェクトを配置するための垂直な角度条件
- **middle angle:** 2つ目のタイプのオブジェクトを配置するための垂直な角度条件

Create 3 types of object

Create three types of object in order to use it for switching.

三種類のオブジェクトを作る

スイッチするための三種類のオブジェクトを作ります。

Create group based on condition

Use Group SOP to set group on each point on an input geometry based on a verticalness angle using Keep by Normals option.

条件に応じてグループを作る

Group SOPのKeep by Normalsオプションを利用してジオメトリ上の各点に、垂直性に応じてグループを付与します。

05-B. Switching with loop

ループを使ってスイッチする



05-switch-B.hiplc

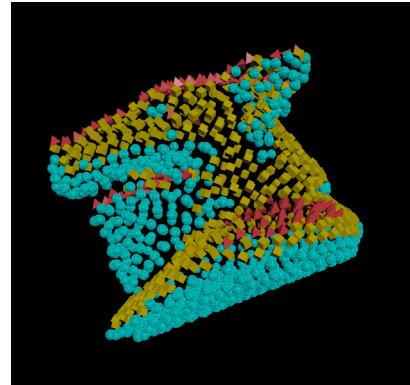
In this example I'm using Switch SOP together with For Each loop. This method can be applied for many other purposes but it is going to be a bit slower than using Copy Stamp method.

Number of points are created on an input geometry and each point will have an attribute calculated with VEX which object to use from three types then use it with Switch SOP inside the loop.

この例ではSwitch SOPとFor Eachループを使っています。

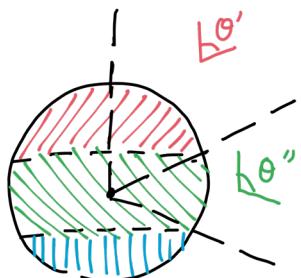
この手法は他の様々な用途に応用することができますが、-

Copy Stampを利用する方法と比べると計算コストが少々高くなります。



インプットのジオメトリに複数のポイントを配置し、各ポイントにどのオブジェクトが使われるかどうかというアトリビュートVEXを使って格納します。そうしたらループの中でこの情報をSwitch SOPで利用します。

①

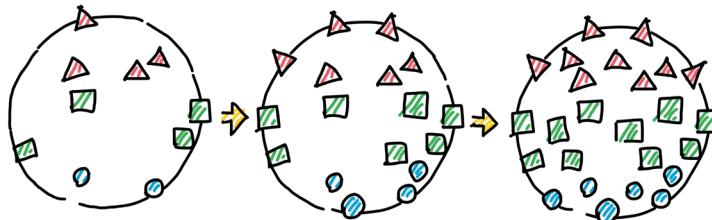


VEX

Use VEX to create three types of group to points on a geometry.

VEXを利用してジオメトリ上のポイントに三つのタイプのグループを作る。

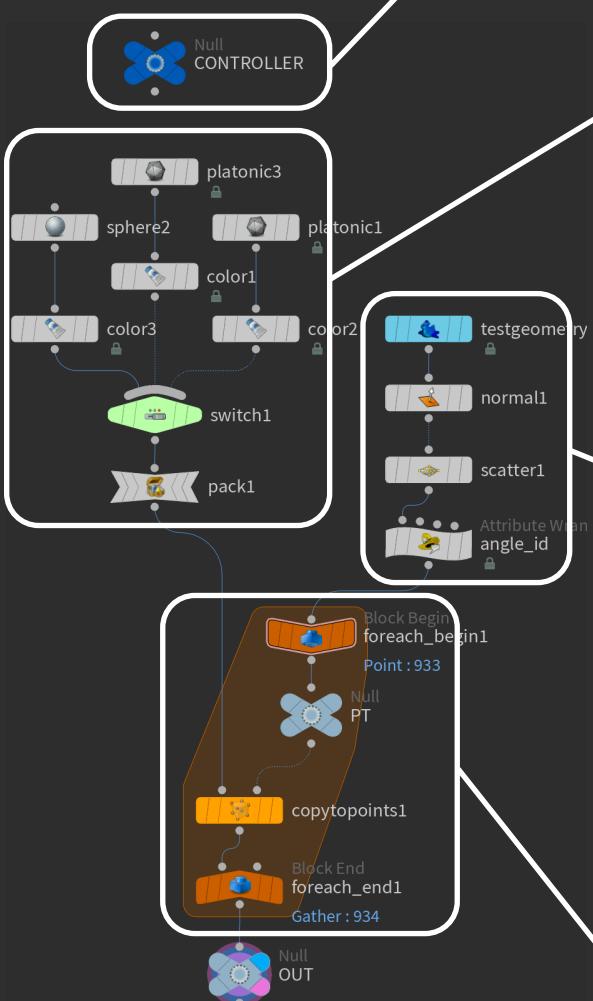
②



FOR EACH LOOP

Use For Each loop to place three types of object one by one on a geometry.

For Eachループを使って一個一個三種類のオブジェクトをジオメトリの上に配置する。



Setup parameters

- **pt_num:** number of points on geometry
- **top_angle:** vertical angle condition for first object type to be placed
- **middle_angle:** vertical angle condition for second object type to be placed

パラメータの設定

- **pt_num:** ジオメトリ上のポイントの数
- **top_angle:** 1つ目のタイプのオブジェクトを配置するための垂直な角度条件
- **middle angle:** 2つ目のタイプのオブジェクトを配置するための垂直な角度条件

Create 3 types of object

Create three types of object in order to use it for switching.

三種類のオブジェクトを作る

スイッチするための三種類のオブジェクトを作ります。

Create points on geometry

Create points on geometry with an angle id calculated from vertical angle.

ジオメトリの上にポイントを作る

ジオメトリの上にポイントを作り、それぞれに垂直角度から計算されたidを付与します。

Place objects on a geometry

Place object on each point on a geometry using loop. The object to place will be chosen using Switch SOP based on point's angle id.

オブジェクトをジオメトリに配置する

ループを使ってジオメトリ上の各ポイントにオブジェクトを配置します。その際、Switch SOPとポイントのid情報を利用して配置するオブジェクトを決定します。

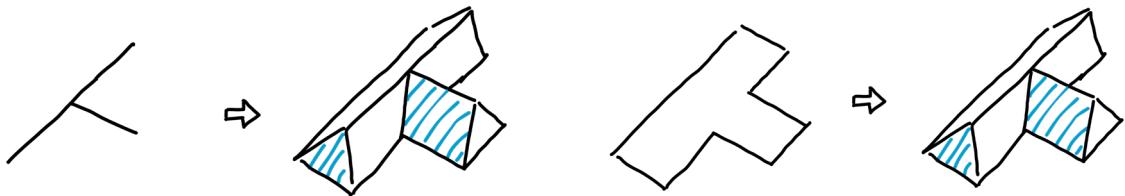
06. Sloped roof 勾配屋根を作る

Create a sloped roof with a width and height parameter. Try to create a roof which could have multiple branches.

幅と高さのパラメータを持った勾配屋根を作ってください。その際、複数の枝分かれを作れるようにしてみてください。

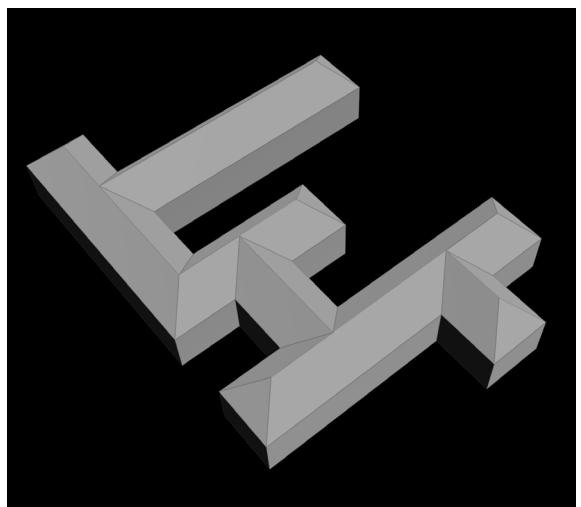
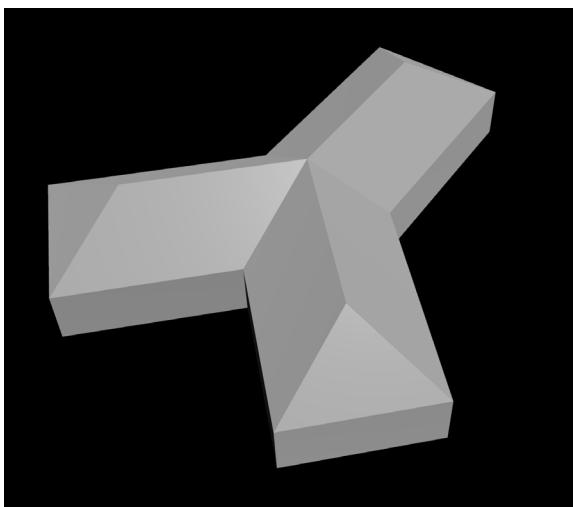
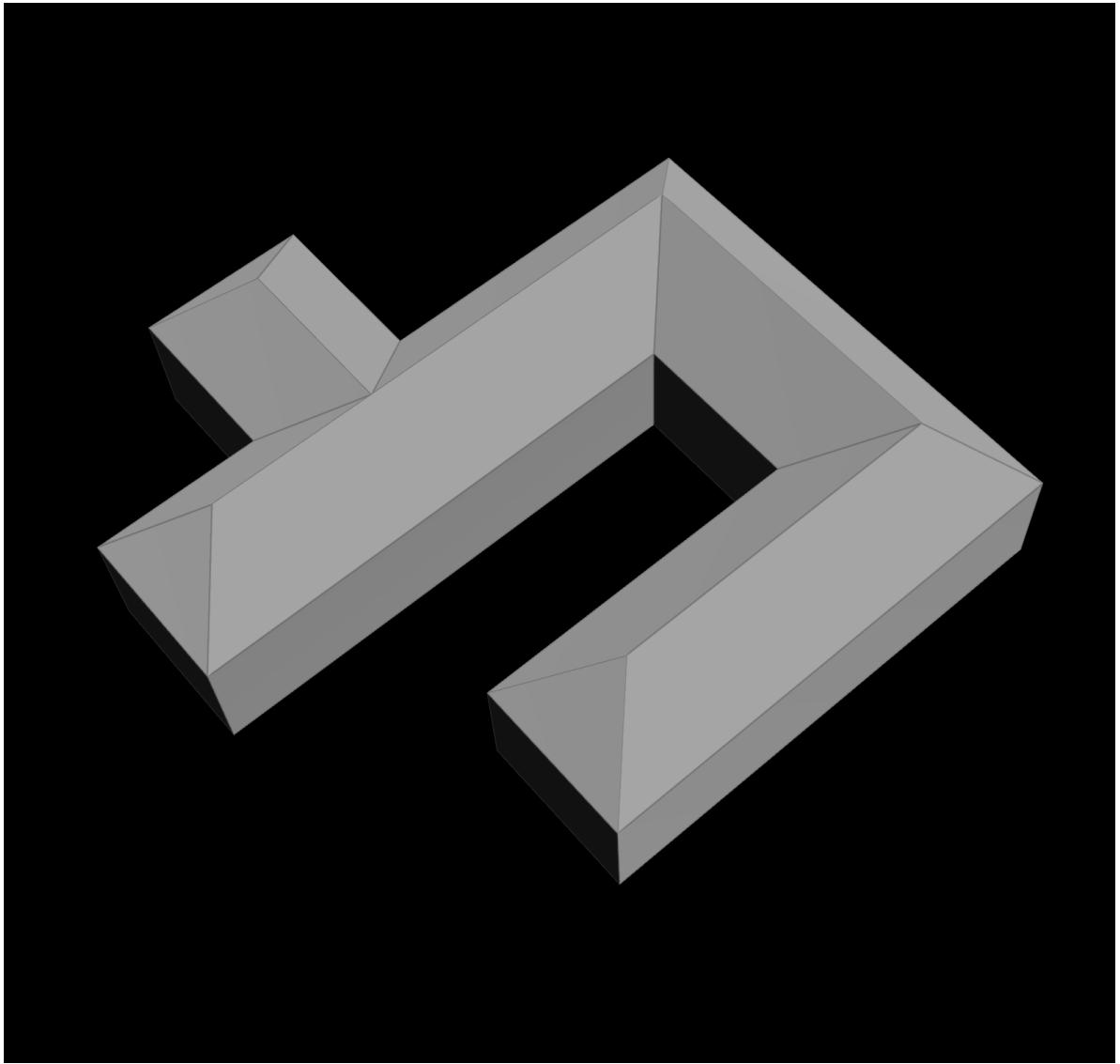
Preferred parameters / 推奨パラメータ:

- Width of a roof / 屋根の幅
- Height of a roof / 屋根の高さ



Create a roof from core lines.
芯のラインから屋根を作る。

Create a roof from an outline.
アウトラインから屋根を作る。



06-A. Create roof with loft

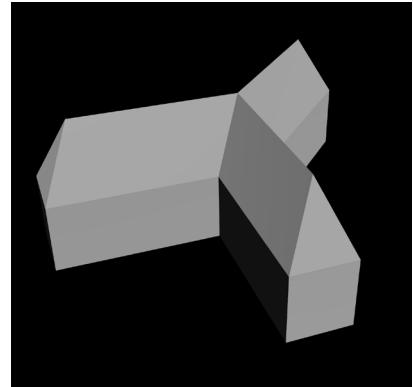
ロフトを使って屋根を作る



06-roof-A.hiplc

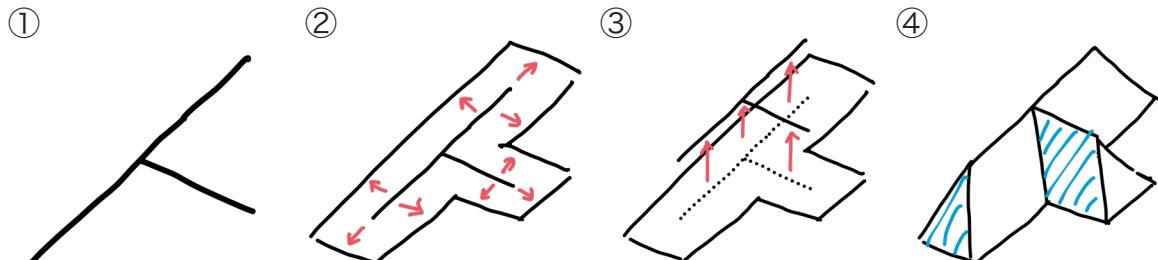
The base idea for this example is to have two curves for the roof, one for the eaves and another one for the ridges and connect them using PolyLoft to create a surface.

This example starts by making core lines of the roof which corresponds to the ridges and uses PolyExpand2D SOP to create both eaves and ridges. Then move the ridge edges upward by a roof height and create a surface from these two edges using PolyLoft SOP.



この例のベースのアイディアは屋根を作るために、PolyLoft SOPを利用するための軒先と棟の二種類の曲線を用意することです。

まず棟に対応するコアのラインを用意し、そのラインに対してPolyExpand2D SOPを使ってPolyLoftで使えるような形で棟と軒先のエッジを作ります。そして棟のエッジを屋根の高さ分移動してロフトでサーフェスを作ります。

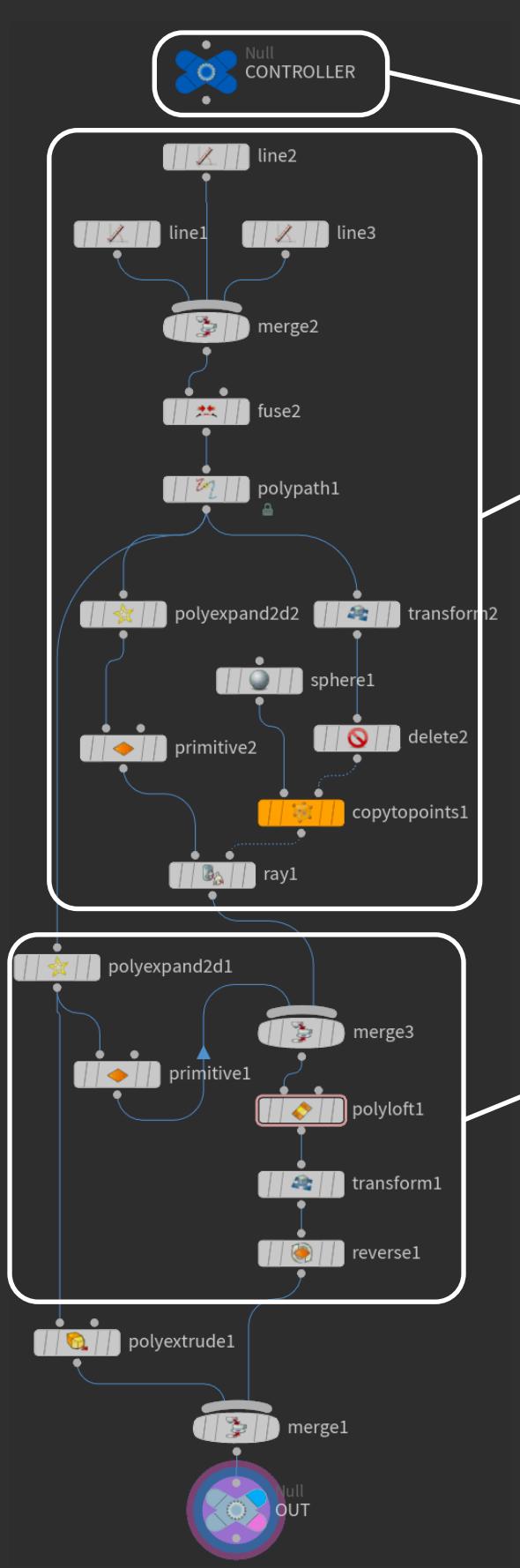


Create core lines.
芯となる線をつくる。

Create an outline
from core lines using
PolyExpand2D SOP.
PolyExpand2D SOPを
使って芯からアウトライン
を作る。

Raise up core lines.
芯の線を上に持ち上げる。

Use PolyLoft SOP to
create roof using the
core lines and the
outline.
PolyLoft SOPを使って
芯の線とアウトラインから
屋根を作る。



Setup parameters

- **height:** height of roof
- **thickness:** width of roof

パラメータの設定

- **height:** 屋根の高さ
- **thickness:** 屋根の幅

Create ridges

Create ridge edges from core lines using PolyExpand2D which can be used for lofting.

棟を作る

ロフトで使える形の棟のエッジを屋根の芯となるラインからPolyExpand2Dを使って作ります。

Create eaves and complete roof

Create eave edges using another PolyExpand2D SOP and connect them with ridge edges using PolyLoft to create roof surface.

軒先のエッジを作り屋根を完成させる

別のPolyExpand2D SOPを使って軒先のエッジを作り、先に作った棟のエッジと合わせてPolyLoftで屋根のサーフェスを完成させます。

06-B. Create roof by pinching center

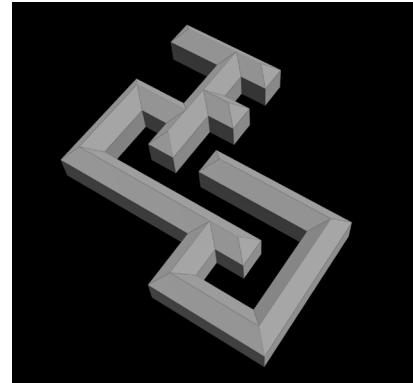


中心をつまんで屋根を作る

06-roof-B.hiplc

This example creates roof from an outline of the roof which corresponds to the roof eaves. The base outline is created by randomly extracting faces from a grid using VEX coding which is a bit complex.

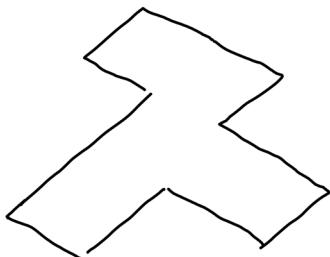
The rest is simple, it's just using PolyExpand2D SOP to create ridge edges and move them up by the height of the roof using VEX to complete the roof surface.



この例では、軒先に対応する屋根のアウトラインから始めて、そこから屋根の棟を作ることで屋根のサーフェスを作ります。この例ではベースのアウトラインはVEXを利用してグリッドの面をランダムに取り出すことで作っていますが、コードの中身自体は少々複雑になっています。

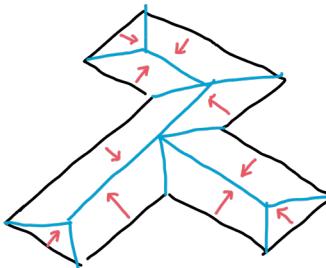
その後の操作は簡単で、アウトラインからPolyExpand2D SOPを利用して棟のエッジを作ります。そのエッジをVEXを利用して屋根の高さ分上に持ち上げることで屋根を完成させることができます。

①



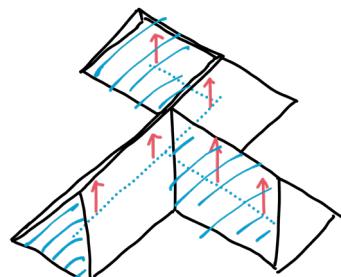
Create an outline from grid.
グリッドからアウトラインを作る。

②

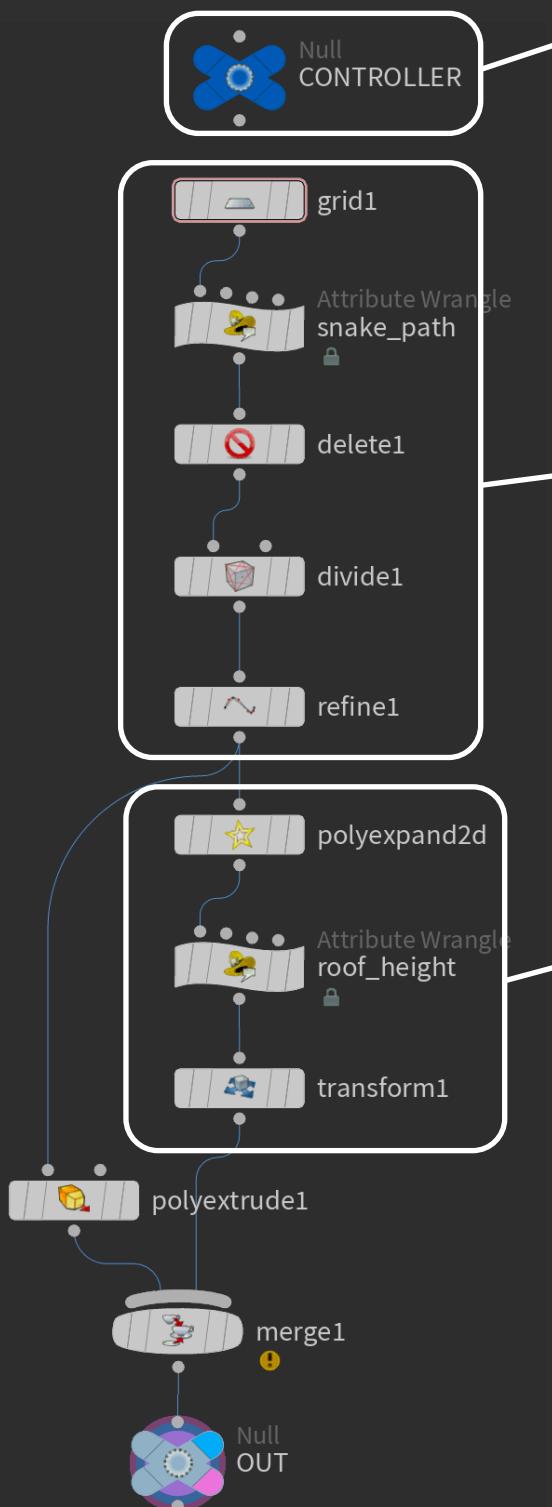


Create core lines using Poly-
Expand2D SOP.
PolyExpand2D SOPを使って屋
根の芯の線を作る。

③



Pinch up core lines to create
roof.
芯の線をつまみ上げて屋根を作
る。



Setup parameters

- **grid:** grid size
- **height:** roof height
- **max length:** max length of roof
- **seed:** random seed for roof

パラメータの設定

- **grid:** グリッドのサイズ
- **height:** 屋根の高さ
- **max length:** 屋根の最大長さ
- **seed:** 屋根のランダムシード

Create roof outline

Create roof outline by randomly extracting faces from grid surface.

屋根のアウトラインを作る

グリッドからランダムに面を取り出すことで屋根のアウトラインを作ります。

Create ridge and pinch up

Use PolyExpand2D SOP to create ridge edges from a roof outline and use VEX to move them up by the roof height to create a roof surface.

棟のエッジを作りつまみ上げる

PolyExpand2D SOPを使って屋根のアウトラインから棟のエッジを作り、VEXを使って屋根の高さ分つまみあげて屋根を作ります。

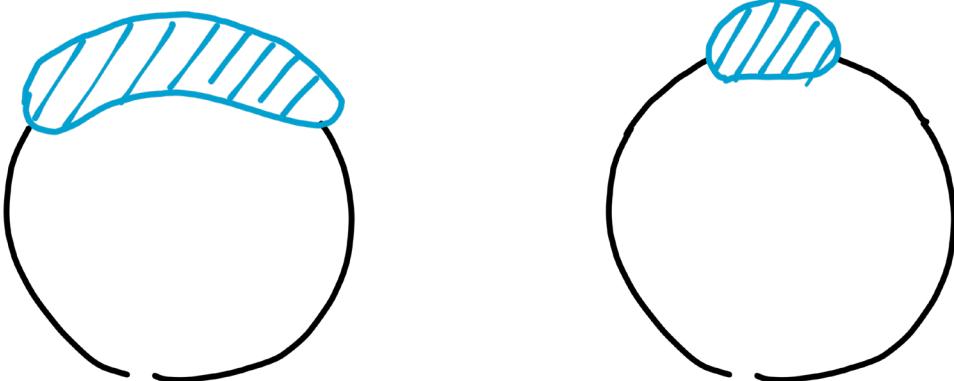
07. **Snow on object** 雪を積もらせる

Create a piled snow on top of any input geometry. Create a parameter so that you can control the snow coverage of the geometry.

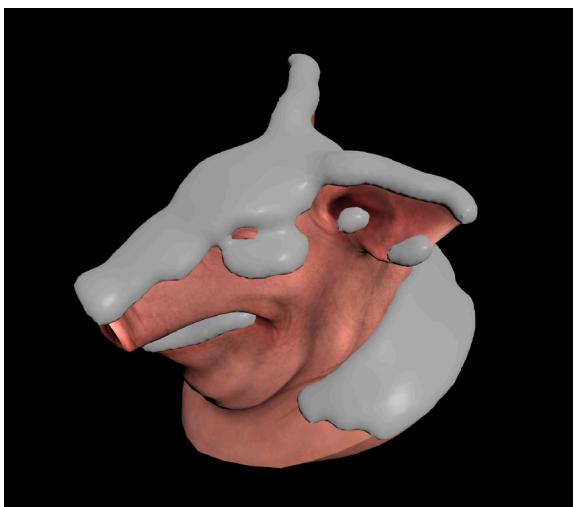
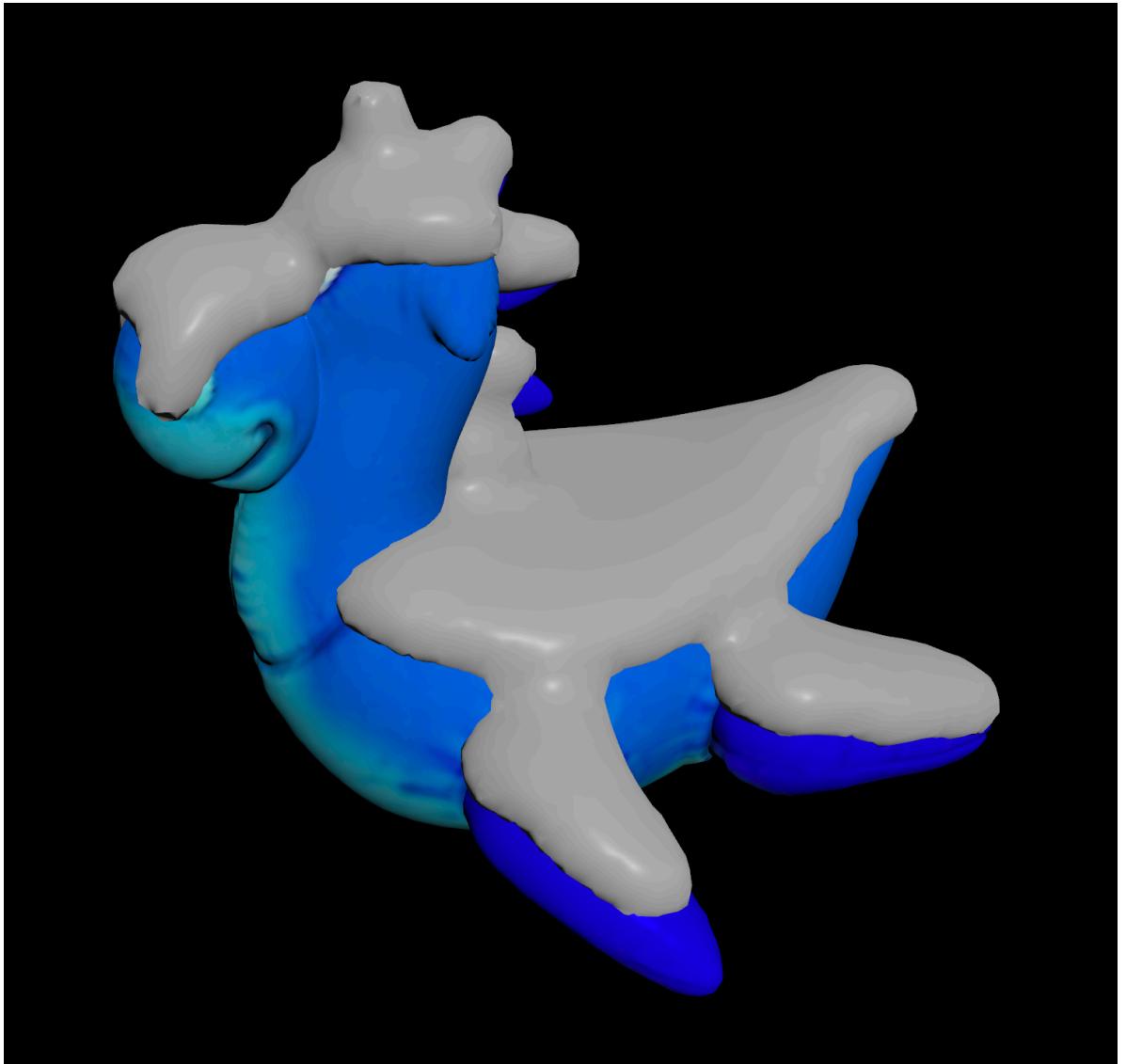
任意のジオメトリの上に、雪を積もらせてください。その際、ジオメトリにおける雪のカバー範囲をコントロールできるパラメータを作ってください。

Preferred parameters / 推奨パラメータ:

- Snow coverage of the geometry / ジオメトリにおける雪のカバー範囲



Pile snow on top of input geometry.
ジオメトリの上に雪を積もらせる。



07-A. Piled snow

積もった雪



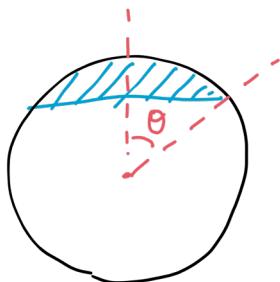
07-snow-A.hiplc

In this example the basic flow is to use Group SOP to capture the top part of an input geometry where you want to have the piled snow. Then use VDB to create an actual snow.

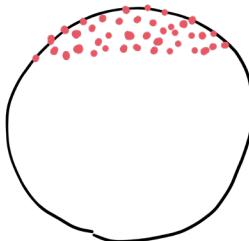
この例の基本的な流れは、まず任意のジオメトリからGroup SOPを使って雪を積もらせたい上部を取得します。その箇所にVDBを利用して実際の雪を作ります。



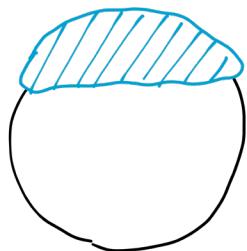
①



②



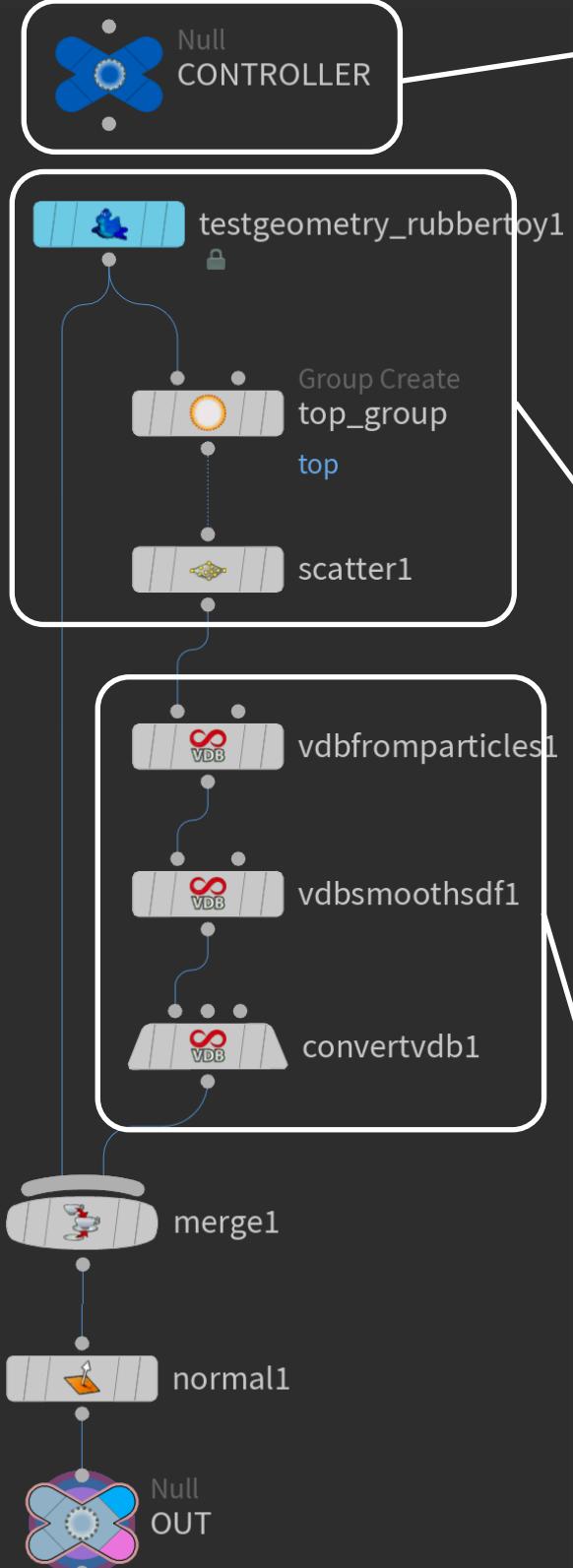
③



Use Group SOP with Keep by Normals option to determine the top part of geometry.
Group SOPとKeep by Normalsのオプションを使ってジオメトリの上部を得る。

Place number of points on top of the geometry using group.
グループを利用してジオメトリの上部に無数のポイントを配置する。

Use VDB from Particles SOP to create thick snow-like geometry from points.
VDB from Particles SOPを使ってポイントから厚い雪のようなジオメトリを作る。



Setup parameters

- **angle**: angle threshold to determine top part of an input geometry
- **density**: density of piled snow

パラメータの設定

- **angle**: ジオメトリの上部を決める角度の閾値
- **density**: 積もらせる雪の密度

Use Group SOP to get top part

Use Group SOP with an angle parameter to get top part of an input geometry then use Scatter SOP to create number of points on this top part.

Group SOPを使って上部を取得する

Group SOPと角度のパラメータを使ってジオメトリの上部を取得します。その上で Scatter SOPを使って無数のポイントを上部に配置します。

Create thick snow

Use VDB from Particles SOP with points on top part of the geometry to create thick snow.

厚みのある雪を作る

VDB from Particles SOPとジオメトリ上部に配置されたポイントを利用して厚みのある雪を作ります。

07-B. Thin snow layer

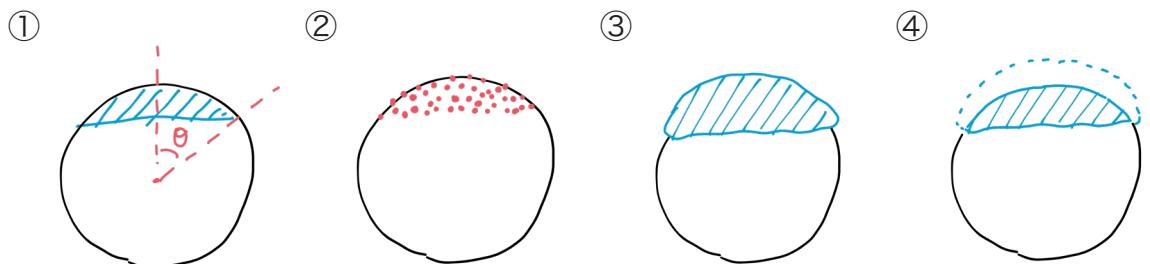
薄い雪の層



07-snow-B.hiplc

Compared to the previous example this one creates a thin layer of snow. It starts by extracting a top part of an input geometry is done using VEX and use VDB as last example to create a thick snow. Then use it to extract the top part of the geometry using Boolean SOP.

前の例と比べるとこちらの例は薄い雪の層を作ります。まずは任意のジオメトリの上部をVEXを使って取得し、前の例のようにVDBを利用して厚い雪の層を作ります。その上で、Boolean SOPを使ってジオメトリから薄い雪の層としてのサーフェスを切り取ります。



① Use VEX to determine the top part of geometry.
VEXを使ってジオメトリの上部を得る。

② Place number of points on top of the geometry using group.
グループを利用してジオメトリの上部に無数のポイントを配置する。

③ Use VDB from Particles SOP to create thick snow-like geometry from points.
VDB from Particles SOPを使ってポイントから厚い雪のようなジオメトリを作る。

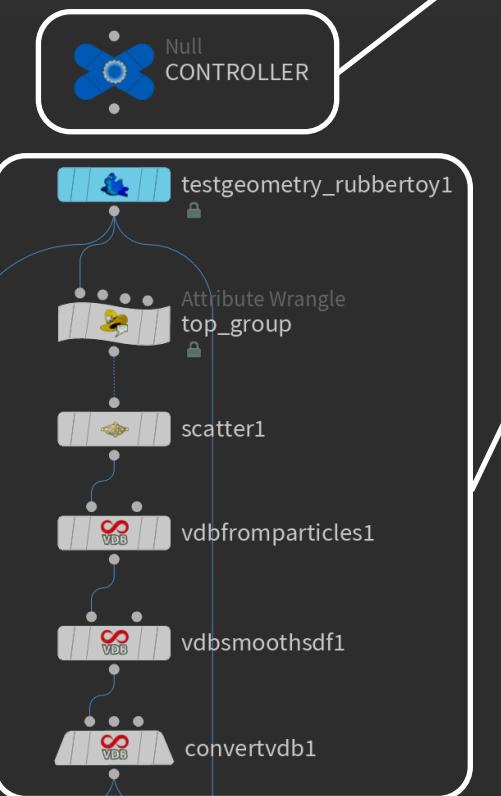
④ Use Boolean SOP with thick snow geometry to trim out a surface to use it as thin snow.
Boolean SOPと厚い雪のジオメトリを使って、ジオメトリから薄い層の雪を取り出す。

Setup parameters

- **angle**: angle threshold to determine top part of an input geometry
- **density**: density of snow

パラメータの設定

- **angle**: ジオメトリの上部を決める角度の閾値
- **density**: 雪の密度

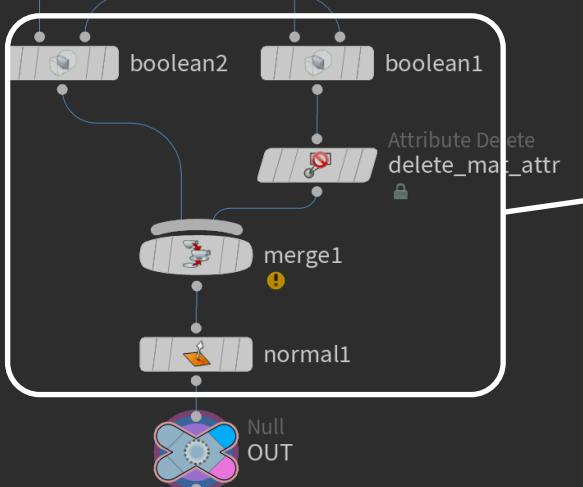


Create thick snow

Use VEX to get the top part of the geometry and place number of points on it. Then use VDB from Particles SOP with points on top part of the geometry to create thick snow.

厚みのある雪を作る

VEXを使ってジオメトリの上部を取得し、そこに無数のポイントを配置します。その上でVDB from Particles SOPとジオメトリ上部に配置されたポイントを利用して厚みのある雪を作ります。



Create thin snow layer

Create a thin layer of snow by using Boolean SOP to get a intersected surface with base geometry and thick snow as inputs.

薄い雪の層を作る

薄い雪の層を、厚みのある雪とベースのジオメトリの交差面をBoolean SOPを使って取り出すことで作ります。

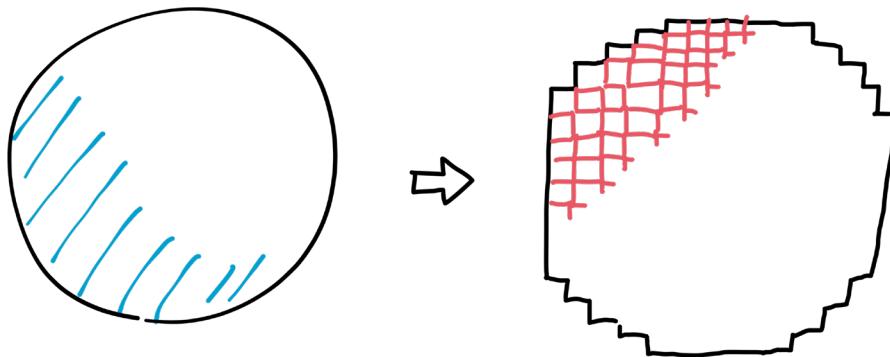
08. **Voxelize object** オブジェクトをボクセル化する

Voxelize any input geometry (convert geometry into collection of boxes like Minecraft) by setting a voxel resolution.

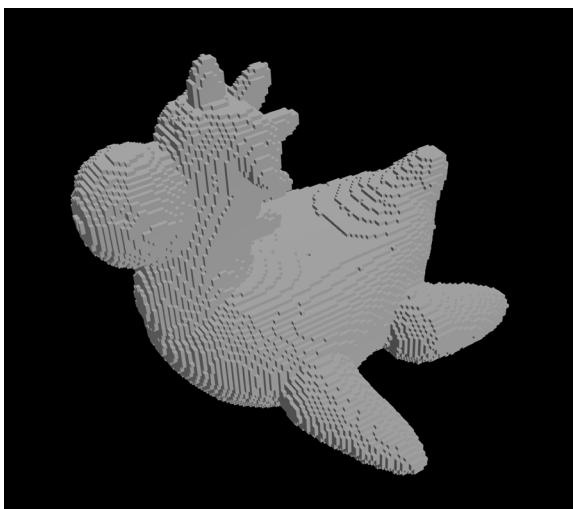
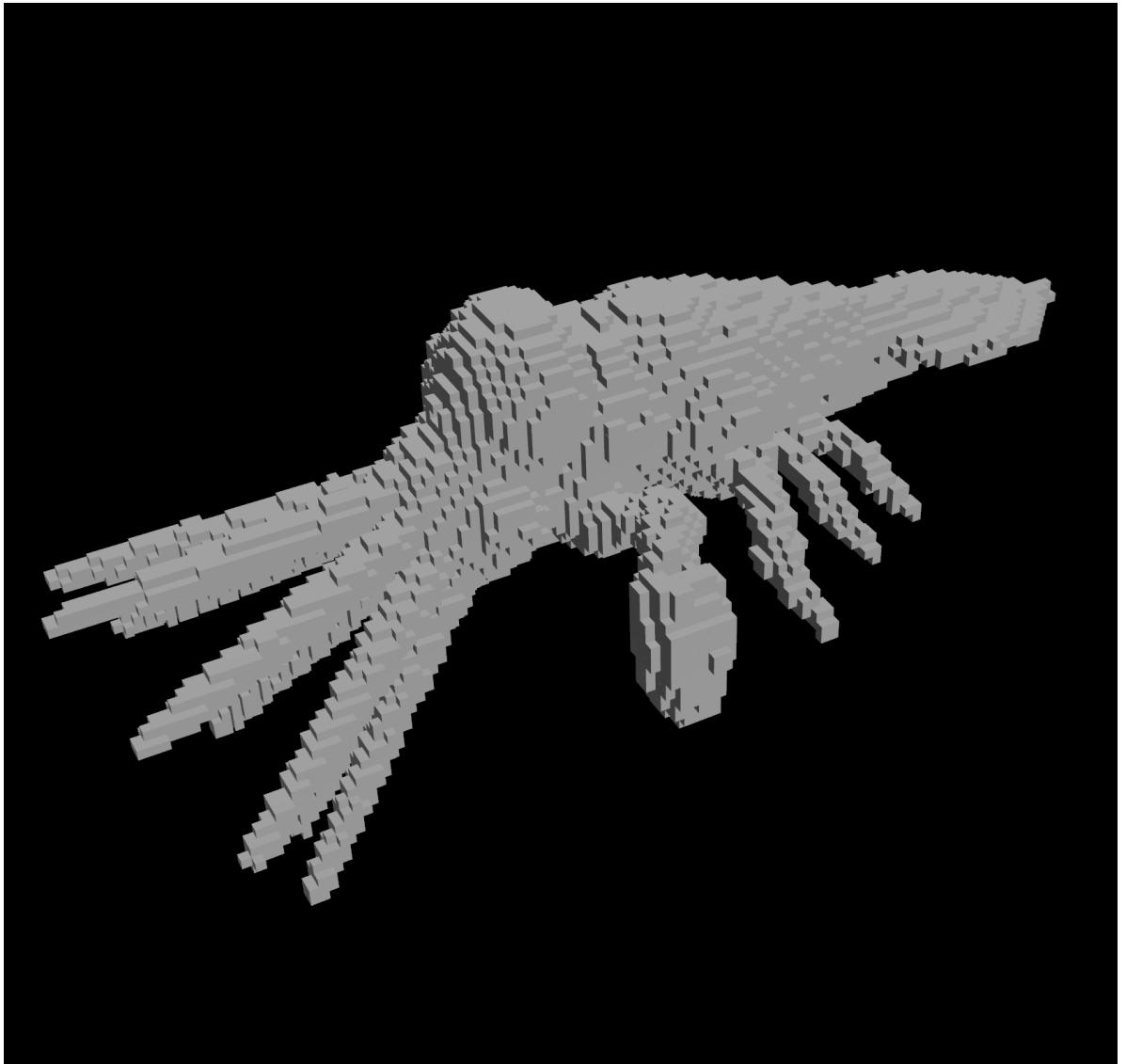
任意のジオメトリを解像度を設定してボクセル化(オブジェクトをボックスの集合として表現)してください。

Preferred parameters / 推奨パラメータ:

- Voxel resolution / ボクセルの解像度



Voxelize input object.
任意のジオメトリをボクセル化する。



08-A. Voxelize using volume

ボリュームを使ってボクセル化する



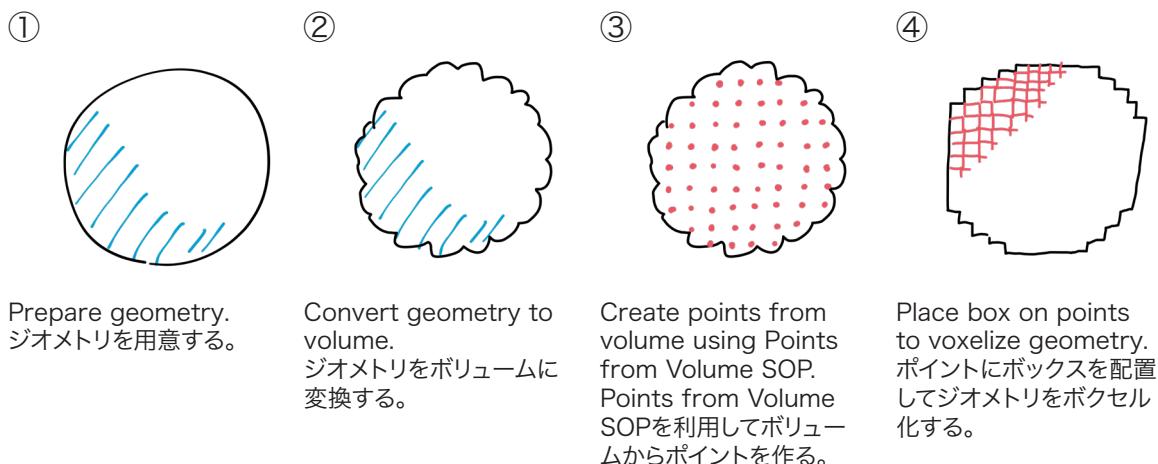
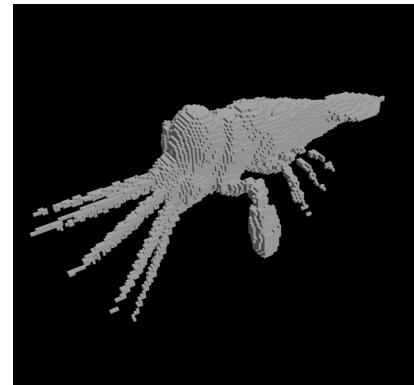
08-voxelize-A.hiplc

This example uses volume to voxelize an input geometry. Specifically this example converts input geometry to volume and uses Points from Volume SOP to extract 3d grid points from a volume and this is one of a simplest way to voxelize an object.

This method is indeed easy but you will have voxels filling up the geometry which makes unnecessary geometries that you cannot actually see.

この例では、ボリュームを利用して任意のジオメトリをボクセル化します。具体的には、この例ではジオメトリをボリュームに変換し、その上でボリュームから3次元のグリッド上に配置されたポイントをPoints from Volume SOPを使って作ります。この方法はオブジェクトをボクセル化する最もシンプルな方法の一つといえると思います。

この手法は簡単ではありますが、ジオメトリの内部にまでボクセルが配置されるため、視覚的に必要のないジオメトリまで生成されるという特徴はあります。

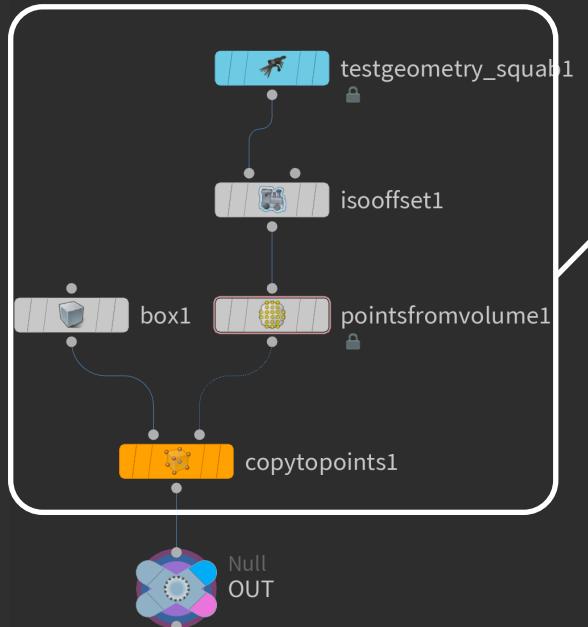


Setup parameters

- **voxel_size**: size of each voxel

パラメータの設定

- **voxel_size**: ボクセルのサイズ



Voxelize geometry

Create a volume from an input geometry and use Points from Volume SOP to create 3d grid points from this volume to use it as a center of voxel box.

ジオメトリをボクセル化する

ボリュームを任意のジオメトリから生成し、そのボリュームをPoints from Volume SOPを利用して3次元のポイントのグリッドを作ります。このポイントがボクセルのボックスタブの中心点となります。

08-B. Voxelize by moving points

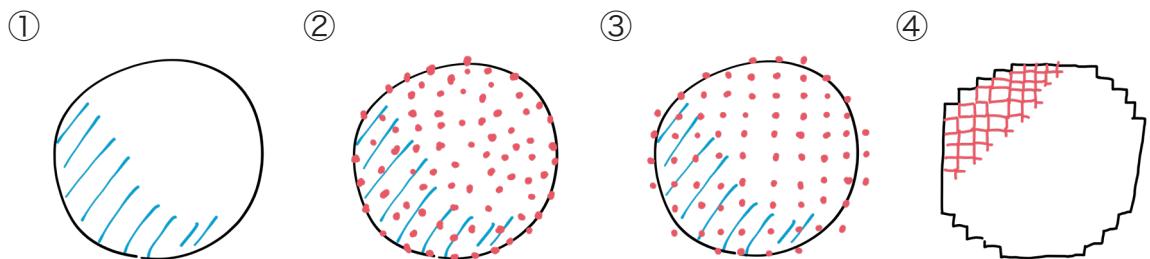
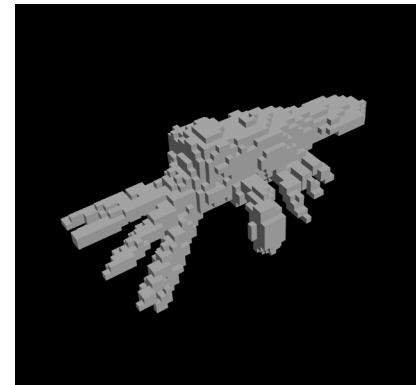
ポイントを移動してボクセル化する



08-voxelize-B.hiplc

This example makes voxels just on the surface of the input geometry using VEX. First create many random points on a geometry using Scatter SOP and then use VEX to move each point to nearest corresponding 3d grid position which can avoid having unseen voxel inside the geometry.

この例ではVEXを利用してボクセルをジオメトリの表面にだけ配置します。まずScatter SOPを利用してランダムなポイントをジオメトリの表面に無数に配置します。その上で、それらのポイントを一番近くの3次元グリッドの頂点の位置に移動します。これによりジオメトリの中に見えないボクセルを配置することを避ける事ができるようになります。



① Prepare geometry.
ジオメトリを用意する。

② Place number of
points using Scatter
SOP.
Scatter SOPを使ってジ
オメトリの上にポイントを
配置する。

③ Using VEX to snap
points to 3d grid.
VEXを利用してポイント
を三次元グリッド上にス
ナップさせる。

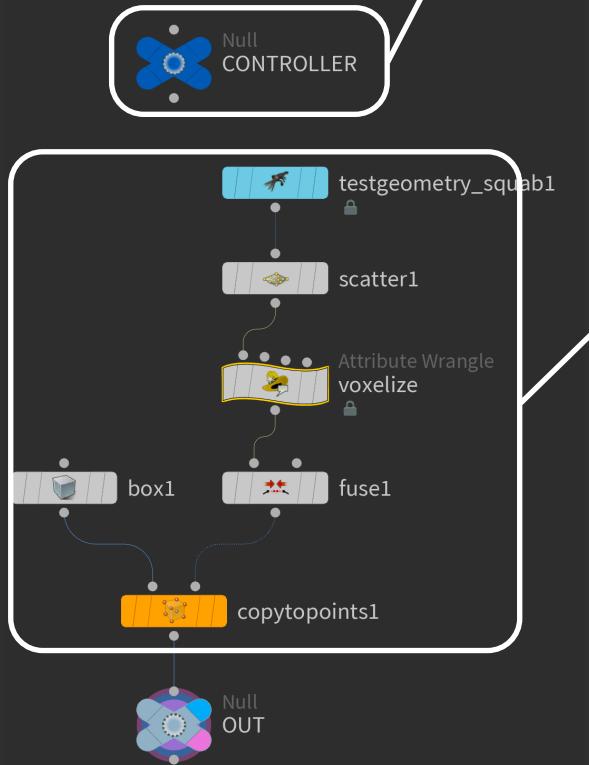
④ Place box on points
to voxelize geometry.
ポイントにボックスを配置
してジオメトリをボクセル
化する。

Setup parameters

- **voxel_size**: size of each voxel

パラメータの設定

- **voxel_size**: ボクセルのサイズ



Voxelize geometry

Create points on a geometry surface using Scatter SOP then move each point position to closest 3d grid point position. Use Fuse SOP to remove duplicated points and this will become the center point for the voxel box.

ジオメトリをボクセル化する

ジオメトリの表面上に無数のポイントを Scatter SOPを利用して作り、そのポイントを一番近い3次元グリッドを構成するポイントに移動します。Fuse SOPを利用して重なっているポイントを一つにまとめ、そのポイントをボクセルのボックスの中心の点として利用します。

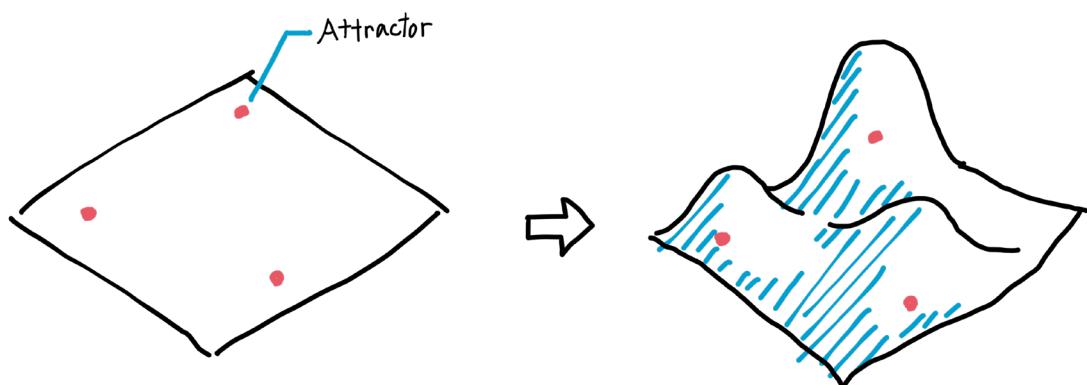
09. Attractor transformation アトラクターで変形する

Create a hexagonal grid and multiple points as attractors. Create a wavy surface from this hexagonal grid based on the attractor points position. If possible create a parameter to control the shape of the wave.

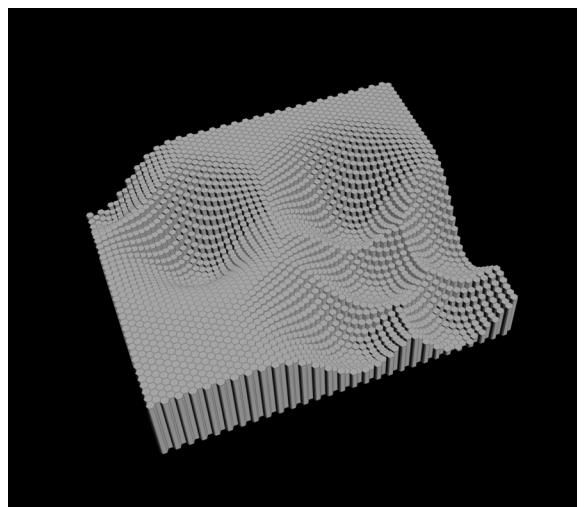
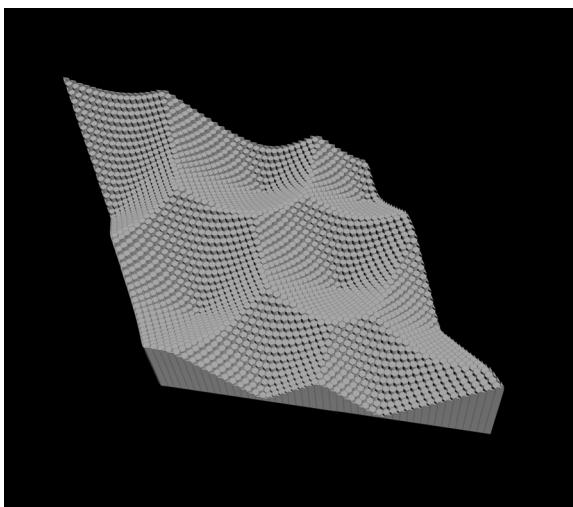
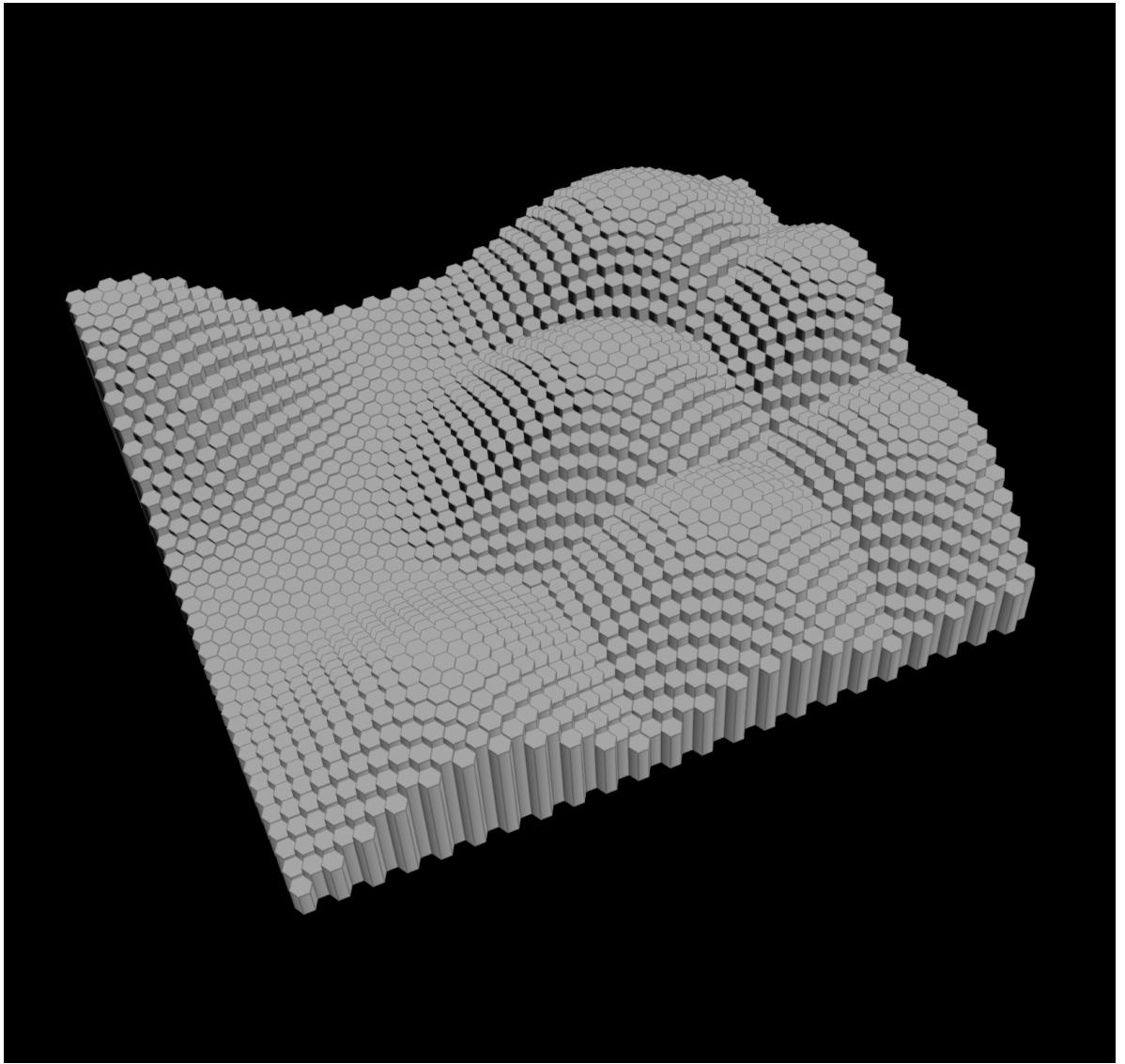
六角形グリッドを作り、複数のアトラクターとなるポイントを作ってください。その上で、アトラクターポイントの位置に応じて六角形グリッドから波打つサーフェスを作ってください。可能であれば、波の形状をコントロールできるパラメータを作ってください。

Preferred parameters / 推奨パラメータ:

- Number of attractor points / アトラクターポイントの数
- Detail of wave shape / 波形状の詳細



Create wavy surface using number of attractor points.
複数のアトラクターポイントから波打つサーフェスを作る。



09-A. Transformation with loop

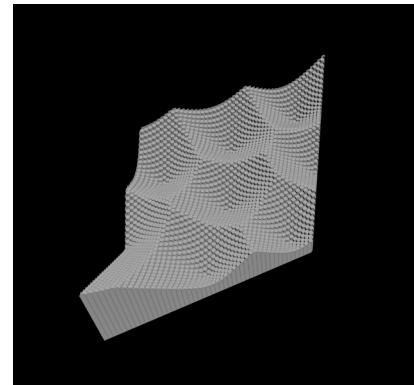
ループを使って変形する



09-attractor-A.hiplc

In order to transform objects using attractor points it is essential to calculate a distance between attractor and objects you want to transform. In this example a Ray SOP is used for each object to transform to calculate the shortest distance between the object and attractor points.

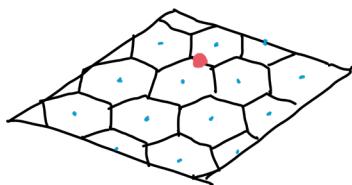
After having a distance this example uses For Each loop to extrude the object using calculated distance as an extrusion height.



アトラクターのポイントを利用してオブジェクトを変形する上で不可欠なのは、変形するオブジェクトとアトラクターのポイント間の距離がわかっていることです。この例では、Ray SOPを利用して変形したい各オブジェクトと一番近いアトラクターポイント間の距離を計算します。

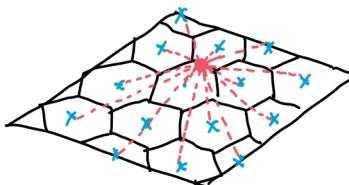
距離が得られたら、For Eachループを使って個々のオブジェクトを距離の値を利用して垂直に押し出します。

①



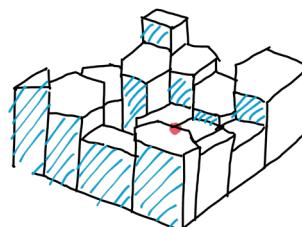
Create hexagonal grid and attractor points.
六角形グリッドとアトラクターポイントを作る。

②

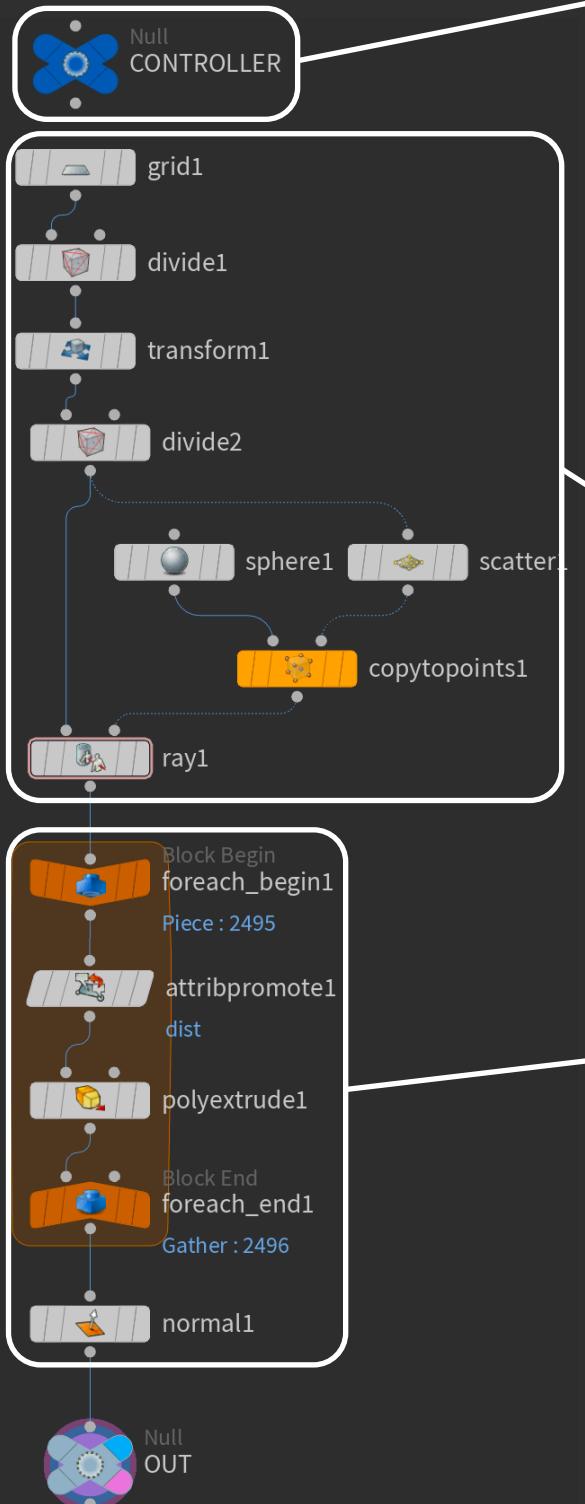


Calculate a distance from each hexagonal grid cell to closest attractor point.
各六角形グリッドのセルから一番近いアトラクターポイントへの距離を計算する。

③



Use distance value to extrude hexagonal grid cells.
距離を値を利用して六角形グリッドのセルを押し出す。



Setup parameters

- **pt_num:** number of attractor points
- **seed:** random seed for attractor points position

パラメータの設定

- **pt_num:** アトラクターポイントの数
- **seed:** アトラクターポイントの位置のためのランダムシード

Calculate distance using Ray SOP

Create a grid and also points as attractors. Use Ray SOP to calculate a distance for each grid face to closest attractor points.

Ray SOPを使って距離を測る

グリッドとアトラクターとしてのポイントを作ります。Ray SOPを使って、個々のグリッドの面から一番近いアトラクターポイントとの距離を測ります。

Use loop to extrude grid faces

Use For Each loop to extrude each grid face with the distance value calculated before using PolyExtrude SOP.

ループを使ってグリッドの面を立ち上げる

For Eachループを使って、グリッドの個々の面を事前に測った距離の値を使ってPolyExtrude SOPで押し出します。

09-B. Transformation with ramp

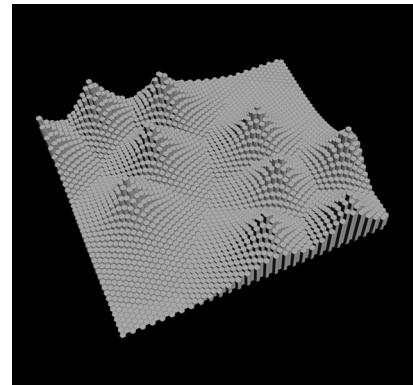


Rampを使って変形する

09-attractor-B.hiplc

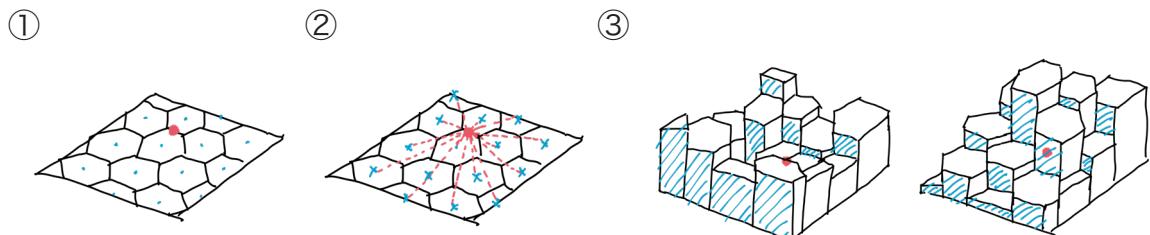
To do an attractor transformation more flexible it's a good idea to use VEX for calculate and modify the distance value between each grid face to closest attractor points. Especially by having ramp parameter you can control the detail of shape of the extruded mountain.

Using VEX you can also prevent using For Each loop for extruding a face which makes the whole process much faster.



アトラクターによる変形をよりフレキシブルに行うためにグリッドの各面からアトラクターポイントへの最短距離を測ったり、その値を変更したりする際、VEXを使うのが便利です。特にランプパラメータを追加することで押し出し形状の詳細をコントロールすることが可能となります。

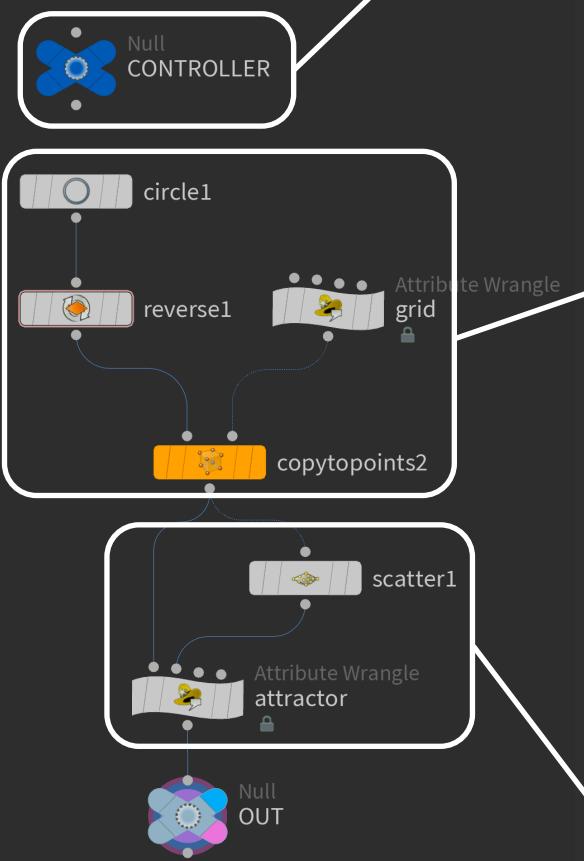
VEXを使うことでFor Eachを介してやっていた押し出し(Extrusion)をVEX単体で行うことができるようになります。これによりプロセス自体を大きく高速化することができます。



Create hexagonal grid and attractor points.
六角形グリッドとアトラクターポイントを作る。

Calculate a distance from each hexagonal grid cell to closest attractor point.
各六角形グリッドのセルから一番近いアトラクターポイントへの距離を計算する。

Use distance value to extrude hexagonal grid cells. Use ramp parameter to control the shape of the wave.
距離を値を利用して六角形グリッドのセルを押し出す。Rampパラメータを利用して波の形状をコントロールする。



Setup parameters

- **cell:** number of grid cell
- **pt_num:** number of attractor points
- **seed:** random seed for attractor points position
- **max_dist:** attractor affection distance
- **height:** max height of extrusion

パラメータの設定

- **cell:** グリッドのセルの数
- **pt_num:** アトラクターポイントの数
- **seed:** アトラクターポイントの位置のためのランダムシード
- **max_dist:** アトラクターの影響範囲
- **height:** 最大立ち上げ高さ

Create a hexagonal grid

Create a hexagonal grid surface by copying hexagonal shape to a hexagonal grid points made with VEX.

六角形グリッドを作る

六角形グリッドサーフェスを、ベースの六角形の面をVEXを使って作った六角形グリッドのポイントの位置にコピーして作ります。

Extrude grid faces with attractors

Use Scatter SOP to create specific number of attractor points on a grid surface. Then use them to calculate the extrusion height with ramp parameter and actually extrude for each grid face with VEX.

アトラクターを使ってグリッドを立ち上げる

Scatter SOPを使って指定の数のアトラクターポイントをグリッド面の上に作る。その上で、VEXを使ってグリッドの各面を立ち上げる高さをランプパラメータを利用して計算し、押し出しまで行います。

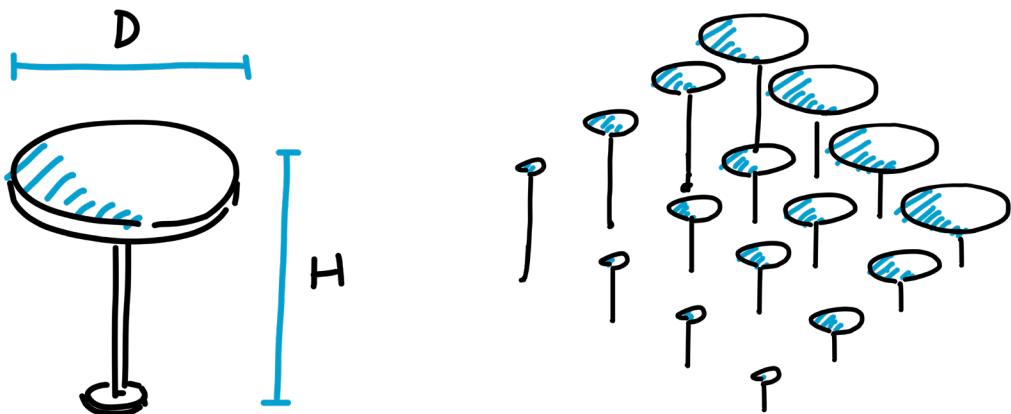
10. Variation layout on a grid グリッド上にバリエーションを配置する

Create a parametric round table which you can change the radius and the height and place it on a 2d grid. Then gradually change the radius parameter of each table based on one axis and change the height parameter based on another axis.

半径と高さをパラメトリックに変えられるラウンドテーブルを作ってください。その上で、二次元グリッド上にテーブルを配置し、一つの軸に沿ってテーブルの半径を変え、もう一つの軸に沿って高さを変化させてください。

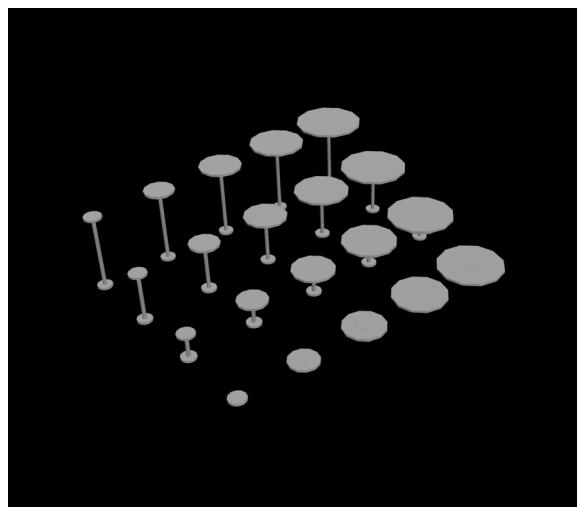
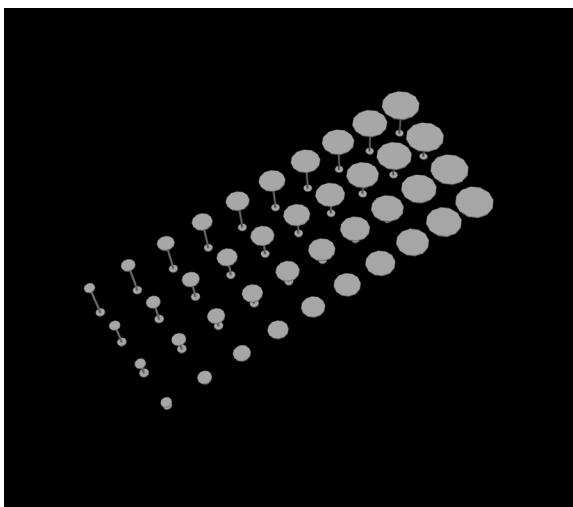
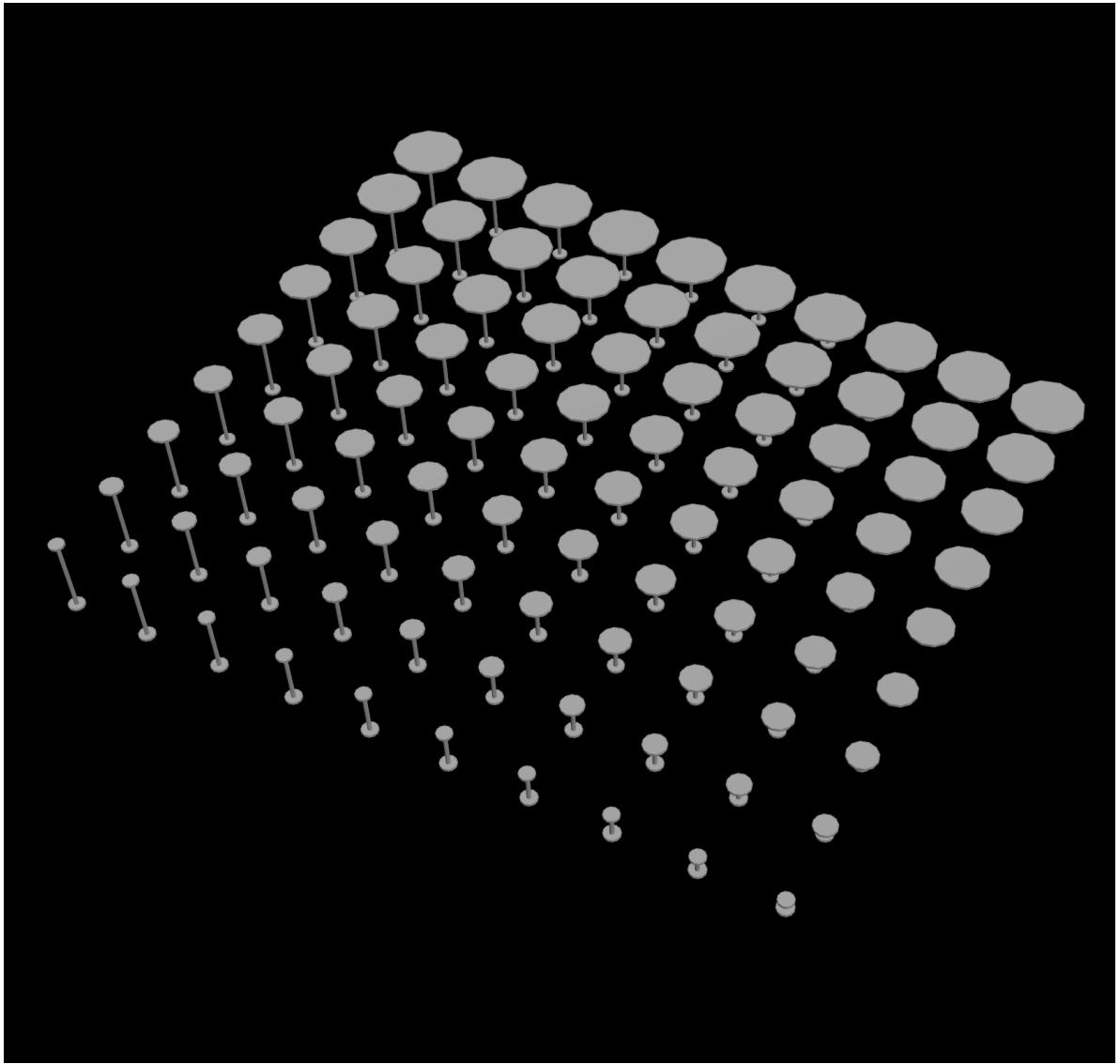
Preferred parameters / 推奨パラメータ:

- 2D grid size and cell number / 二次元グリッドの大きさとセルの数
- Minimum and maximum radius of the table / テーブルの最小最大の半径
- Minimum and maximum height of the table / テーブルの最小最大の高さ



Create parametric round table.
パラメトリックな丸テーブルを作る。

Layout the parametric table on a 2d grid and change parameters based on 2 axis.
パラメトリックなテーブルを二次元のテーブルに配置し、二軸に応じてパラメータを変える。



10-A. Layout using nested loop

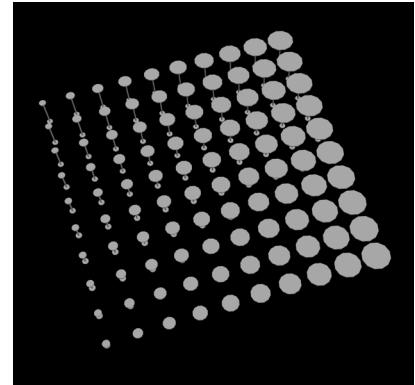
二重ループを使ってレイアウトする



10-grid_variation-A.hiplc

Layout parametric object on 2d grid with gradually changing parameters can be rather easily done using nested For Each loop.

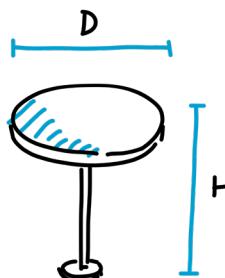
In this example's case a round table with two parameters is prepared. Then two For Each Number loop is used to gradually change the table's two parameters. At each iteration you can calculate the parameter for the table using the two iteration values from the loop.



パラメトリックなオブジェクトを二次元グリッド上にレイアウトし、グラデーションナルにパラメータを変更することは二重のFor Eachループを使うことで比較的簡単に行うことができます。

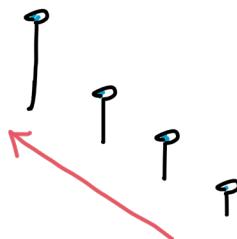
この例の場合、2つのパラメータを持った丸テーブルを用意します。その上で、2つのFor Each Numberループを使って、各ループの際に得られる何回目のループかという値を利用してグラデーションナルにパラメータを変化させます。

①



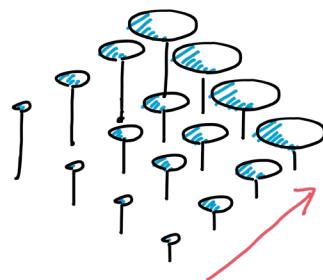
Create parametric round table.
パラメトリックな丸テーブルを作成する。

②

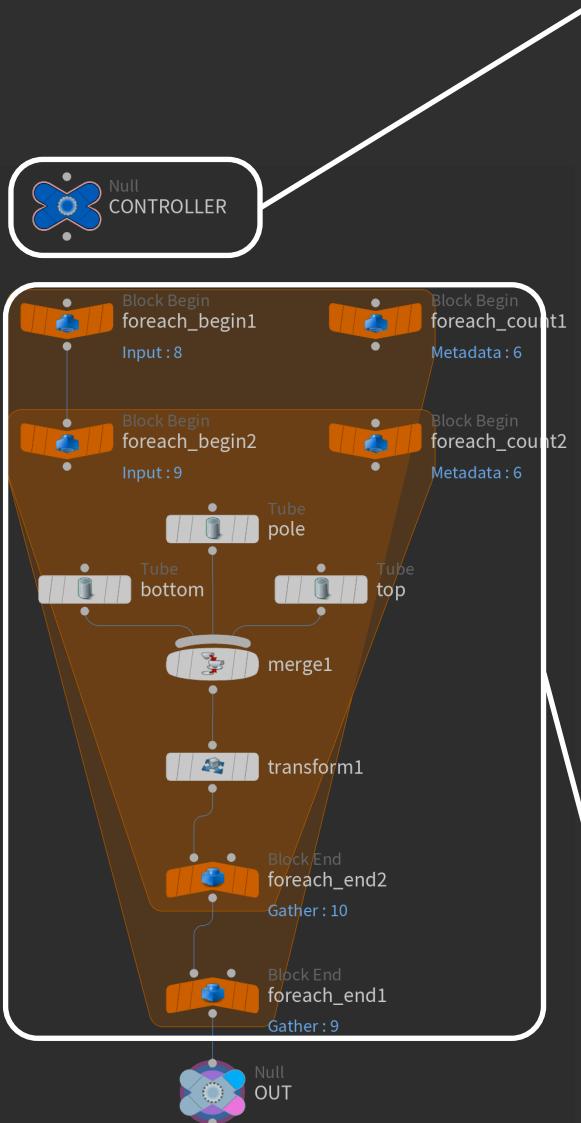


Use For Each loop to layout table on one axis by changing height parameter.
For Eachループを使って一軸に高さのパラメータを変更して机を配置する。

③



Use another For Each loop to layout table on another axis by changing size parameter.
別のFor Eachループを使ってもう一つの軸にサイズのパラメータを変更して机を配置する。



Setup parameters

- **height_num**: number of iterations for height parameter
- **size_num**: number of iterations for size parameter
- **min_height**: minimum height
- **max_height**: maximum height
- **min_size**: minimum size
- **max_size**: maximum size
- **spacing**: spacing value for 2d grid

パラメータの設定

- **height_num**: 高さのパラメータ変更のためのループの回数
- **size_num**: サイズのパラメータ変更のためのループの回数
- **min_height**: 最小高さ
- **max_height**: 最大高さ
- **min_size**: 最小サイズ
- **max_size**: 最大サイズ
- **spacing**: 二次元グリッドの間隔距離

Use nested loop to layout

Use two For Each Number loop nested so that you can have $x * y$ iterations. At each iteration use iteration value from each For Each loop to use it for parameters for round table.

二重ループでレイアウトする

$x * y$ の数だけのループが得られるようにふたつのFor Each Numberループを使って二重ループを作ります。個々のループ処理の際、何回目のループかという値を利用して丸テーブルのパラメータを変更します。

10-B. Layout using attributes

アトリビュートを使ってレイアウトする



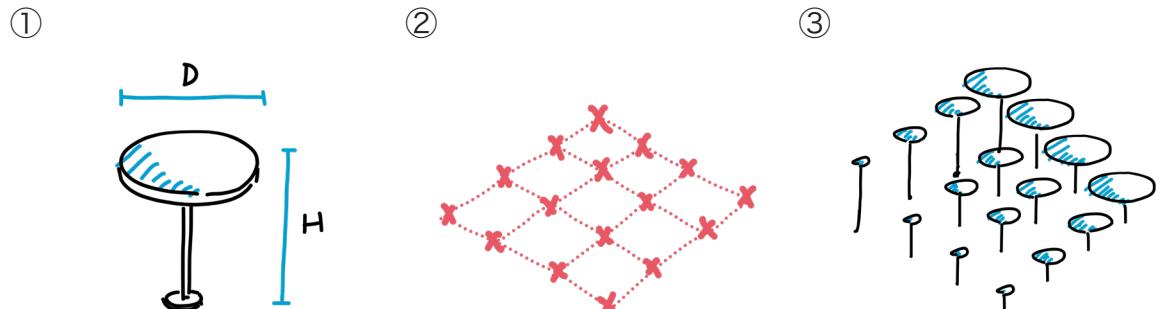
10-grid_variation-B.hiplc

This example uses attributes created by VEX instead of using nested For Each loop to do a gradually changing 2d grid layout which makes the whole process much faster.

In this case the parameter I want to gradually change can be expressed as a scale value, so setting each grid point with scale vector attribute will do the job when used with Copy to Points SOP.

この例では二重ループの代わりにVEXによってアトリビュートを作ることで二次元グリッド上にレイアウトする手法を使っています。この手法を使うと全体のプロセスが前の例よりもっと高速になります。

今回の場合、変化させたいパラメータはスケール値で表現することができるので、個々のグリッドのポイントにscaleというベクトルアトリビュートを設定し、Copy to Points SOPを利用してグラデーション的にパラメータが変化するレイアウトを作ることができます。



Create parametric round table.
パラメトリックな丸テーブルを作る。

Create 2d grid and set scale attribute to each point based on it's position.
二次元グリッドを作り、各ポイントにその位置に応じてscaleのアトリビュートを設定する。

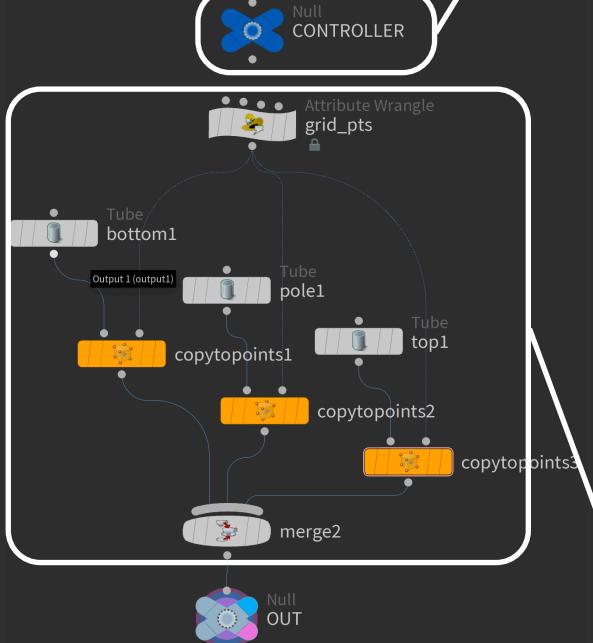
Use Copy to Points SOP to place round table on each point with scale attribute to change table shape.
Copy to Points SOPを利用して、scaleアトリビュートが設定された各ポイントにテーブルを配置し、テーブルの形状を変形する。

Setup parameters

- **height_num:** number of iterations for height parameter
- **size_num:** number of iterations for size parameter
- **min_height:** minimum height
- **max_height:** maximum height
- **min_size:** minimum size
- **max_size:** maximum size
- **spacing:** spacing value for 2d grid

パラメータの設定

- **height_num:** 高さのパラメータ変更のためのループの回数
- **size_num:** サイズのパラメータ変更のためのループの回数
- **min_height:** 最小高さ
- **max_height:** 最大高さ
- **min_size:** 最小サイズ
- **max_size:** 最大サイズ
- **spacing:** 二次元グリッドの間隔距離



Create layout using attributes

Create grid points with scale attributes based on the position. Then use Copy to Points SOP to place tube geometry scaled using this attribute to create a round table.

アトリビュートを利用してレイアウトする
グリッドを作り、個々のグリッドの点にscaleのアトリビュートを設定します。その上で、- Copy to Points SOPを利用して円柱形状をscaleアトリビュートで変形しつつ配置し、丸テーブルを作ります。

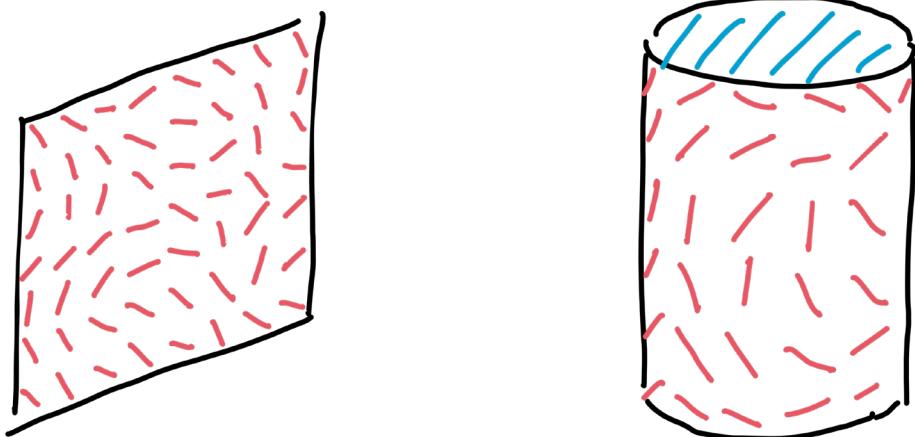
11. Undulated panels 波打つパネルを作る

Place number of panels on any input geometry and rotate each panel to create the overall undulation.

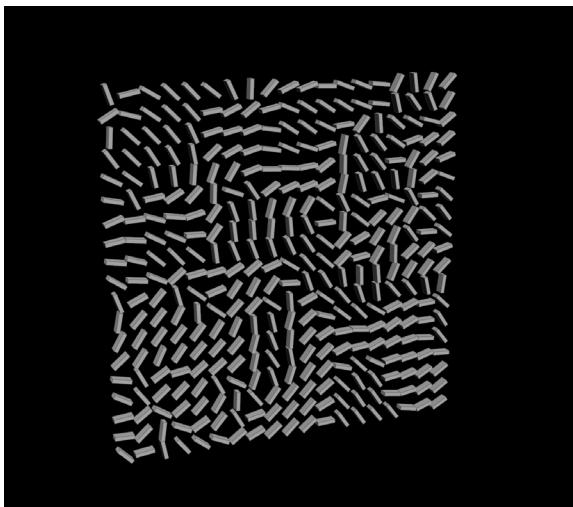
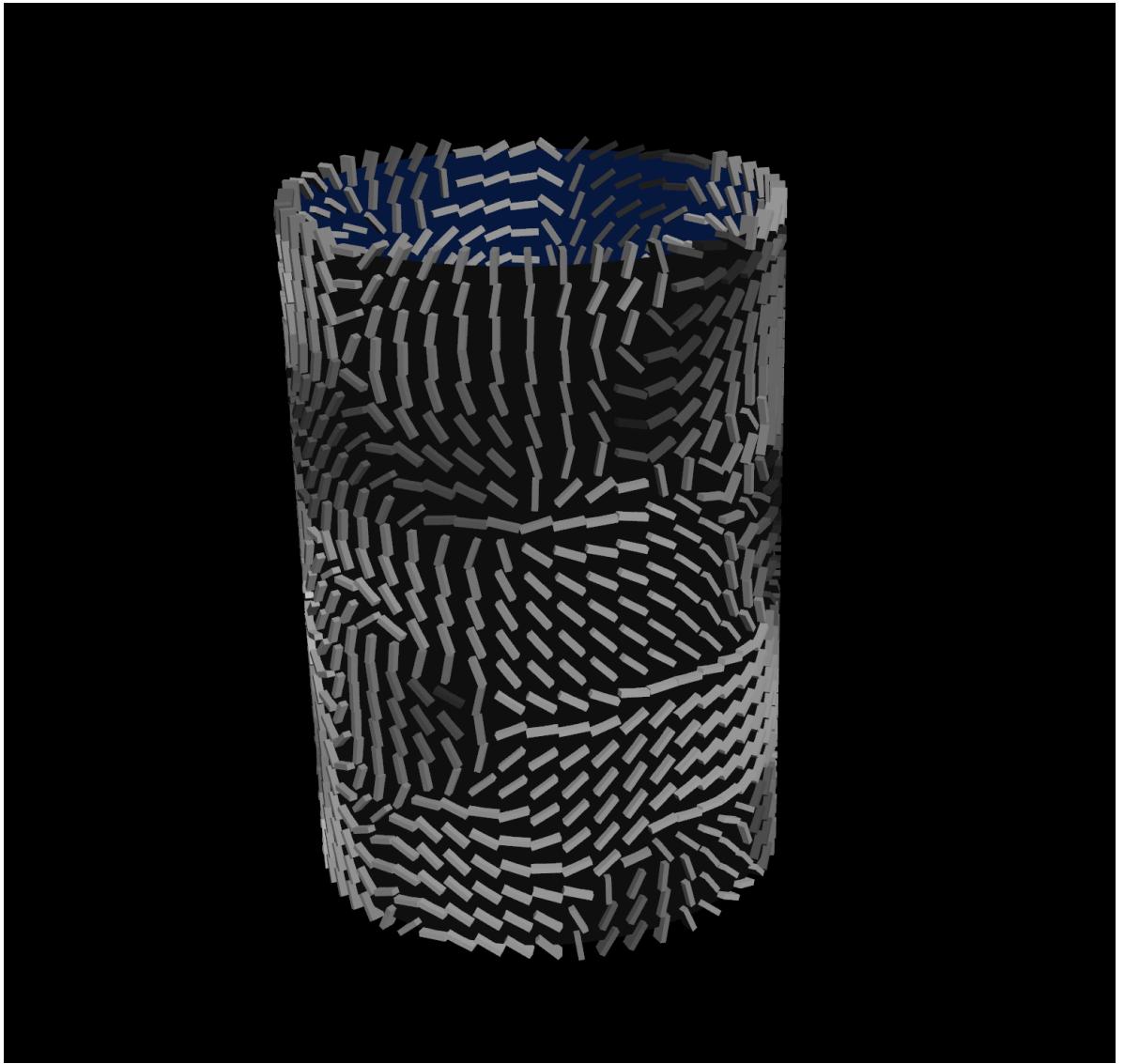
任意のジオメトリに複数のパネルを配置して、それぞれを回転することで全体的なうねりを作ってください。

Preferred parameters / 推奨パラメータ:

- Undulation smoothness / うねりの滑らかさ



Create undulated panels on a surface.
サーフェスの上に波打つようにパネルを配置する。



11-A. Place panels on flat plane

平面の上にパネルを配置する



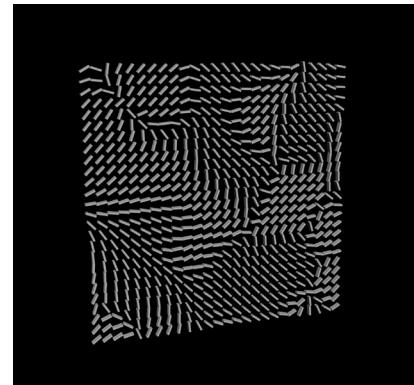
11-undulated_panel-A.hiplc

Making undulated panel can be done in many ways but one of the easiest way to do this without any complex setup is to use Attribute Noise SOP together with Copy to Points SOP.

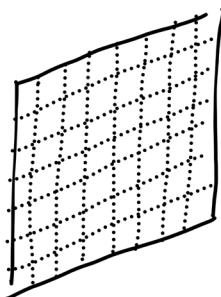
To make it more simple this example creates panels on a flat plane which doesn't need to consider various normal direction.

波打つパネルを作るのには無数の方法が考えられます、複雑な設定なしで最も簡単な方法のひとつとして考えられるのは、Attribute Noise SOPとCopy to Points SOPを利用することです。

またよりシンプルするために、この例ではフラットな平面にパネルを作っています。これによりノーマル方向が一定に決まるのでより簡単に考えることが可能となります。

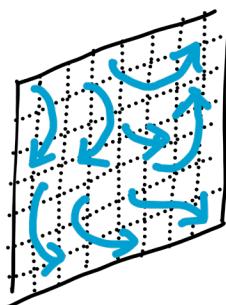


①



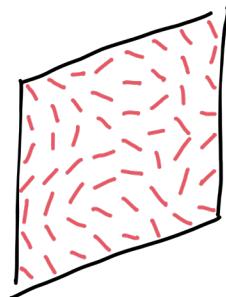
Create a grid plane.
グリッド平面を作る。

②

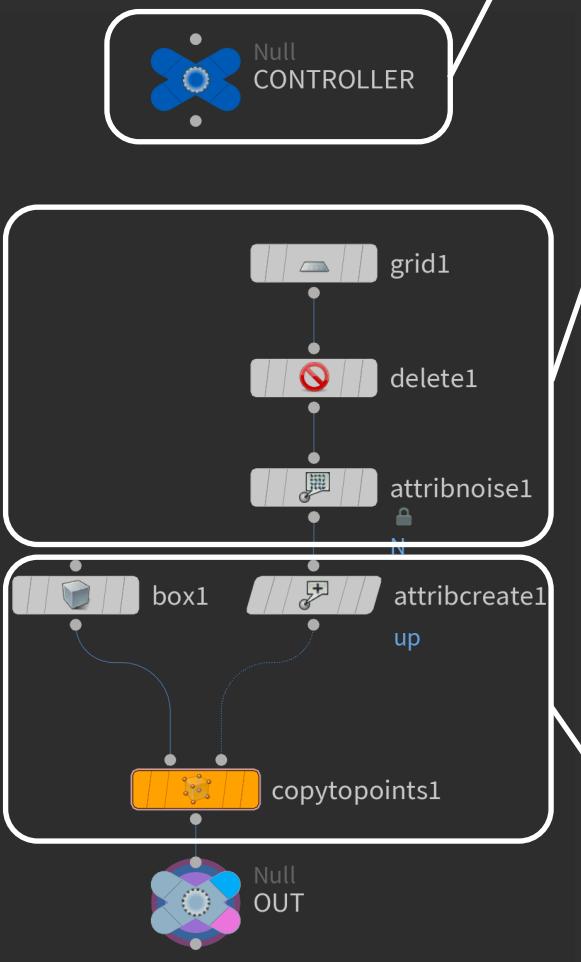


Use Attribute Noise SOP to set random normal on each point.
Attribute Noise SOPを利用して各ポイントにランダムな法線を設定する。

③



Place panel on each point on grid using Copy to Points SOP.
グリッドの各ポイントにCopy to Points SOPを利用してパネルを配置する。



Setup parameters

- **cell_num:** grid cell number
- **noise_scale:** noise function scale value
- **noise_offset:** noise function offset value

パラメータの設定

- **cell_num:** グリッドセルの数
- **noise_scale:** ノイズ関数のスケール値
- **noise_offset:** ノイズ関数のオフセット値

Create noise value for normal

Use Attribute Noise SOP for each point in a grid to create a normal (N) attribute.

法線方向をノイズで作る

Attribute Noise SOPを利用して、グリッドの個々のポイントに法線情報Nをアトリビュートとして設定する。

Place panel on grid points

Use Copy to Points SOP to place panel on grid points. Since each grid point have normal attribute the panel automatically rotates.

グリッドのポイントにパネルを配置する

Copy to Points SOPを利用してパネルをグリッドのポイントに配置します。グリッドの各ポイントにはすでに法線情報がアトリビュートとして入っているため、パネルは自動的にそれに応じて回転します。

11-B. Place panels on any surface

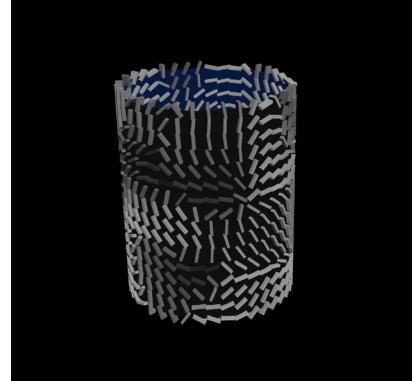
★★★☆

好きなサーフェスにパネルを配置する

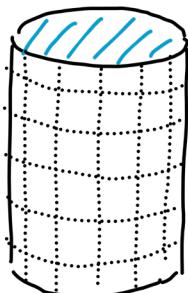
11-undulated_panel-B.hiplc

In order to use a bit more complex shape compared to flat surface to place panels it's a good idea to use VEX to implement noise function which could be more flexible than using Attribute Noise SOP. In this way you can directly set normal vector using VEX. In this example I'm choosing tube shape for base geometry but any other geometry could be used with this method.

フラットなサーフェスよりもより複雑なサーフェスにパネルを配置するには、VEXでノイズ関数を利用するるのがAttribute Noise SOPを利用するよりもよりフレキシブルかと思われます。これにより直接法線のアトリビュートを操作することも可能です。この例では円柱形状をベースのジオメトリとして選択していますが、他の形状でも同じ手法が使えます。

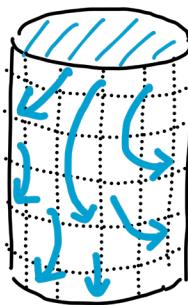


①



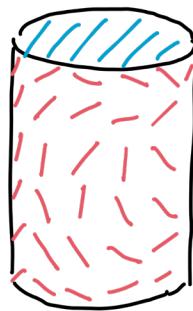
Create a tube geometry.
円柱形状を作る。

②



Use VEX with noise function
to set random normal on
each point.
VEXのnoise関数を利用して各
ポイントにランダムな法線を設定
する。

③



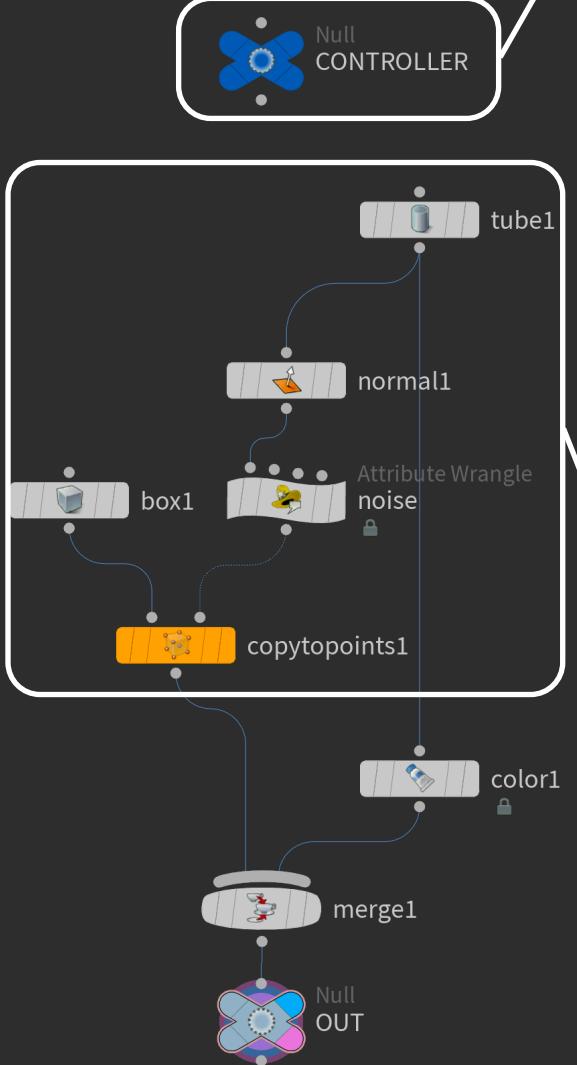
Place panel on each point
on surface using Copy to
Points SOP.
サーフェスの各ポイントにCopy
to Points SOPを利用してパネル
を配置する。

Setup parameters

- **round_num:** number of circle division for tube
- **height_num:** number of height division for tube
- **noise_scale:** noise scale value
- **nosie_offset:** noise offset value

パラメータの設定

- **round_num:** 円柱の円の解像度
- **height_num:** 円柱の高さの解像度
- **noise_scale:** ノイズ関数のスケール値
- **nosie_offset:** ノイズ関数のオフセット値



Use VEX to create normal

Create normal vector using noise function with VEX for each point on tube geometry.

VEXを使って法線ベクトルを作る

VEXを使い、円柱形状の各ポイントに対してノイズ関数を利用して法線方向を設定します。

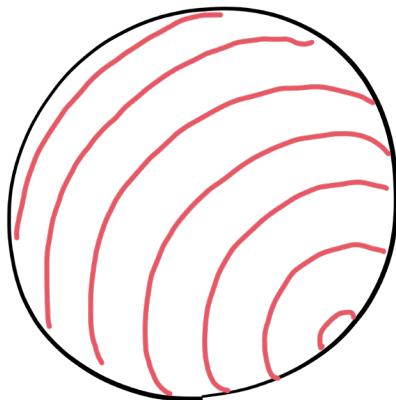
12. Rotational slicing オブジェクトを回転方向で切り出す

Slice sphere object using rotations. Try to slice the object cylindrically (one axis rotation) or spherically (three dimensional rotation).

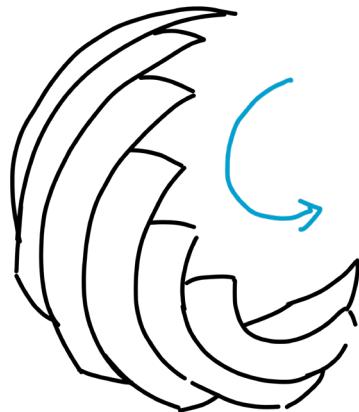
球体を回転を利用してスライスしてください。円柱状にスライス(一軸の回転)か、球体状にスライス(三軸の回転)を試してみてください。

Preferred parameters / 推奨パラメータ:

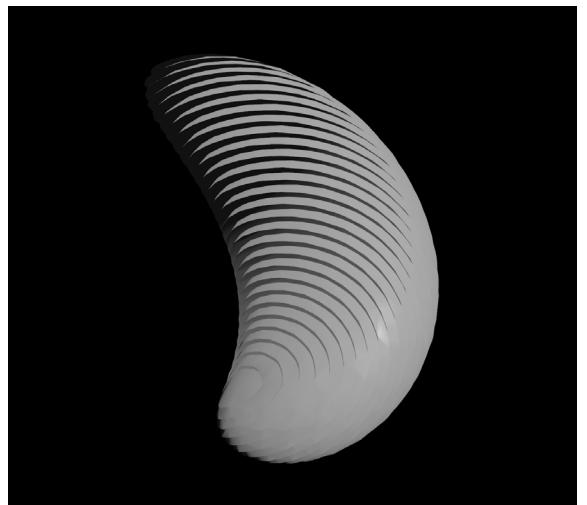
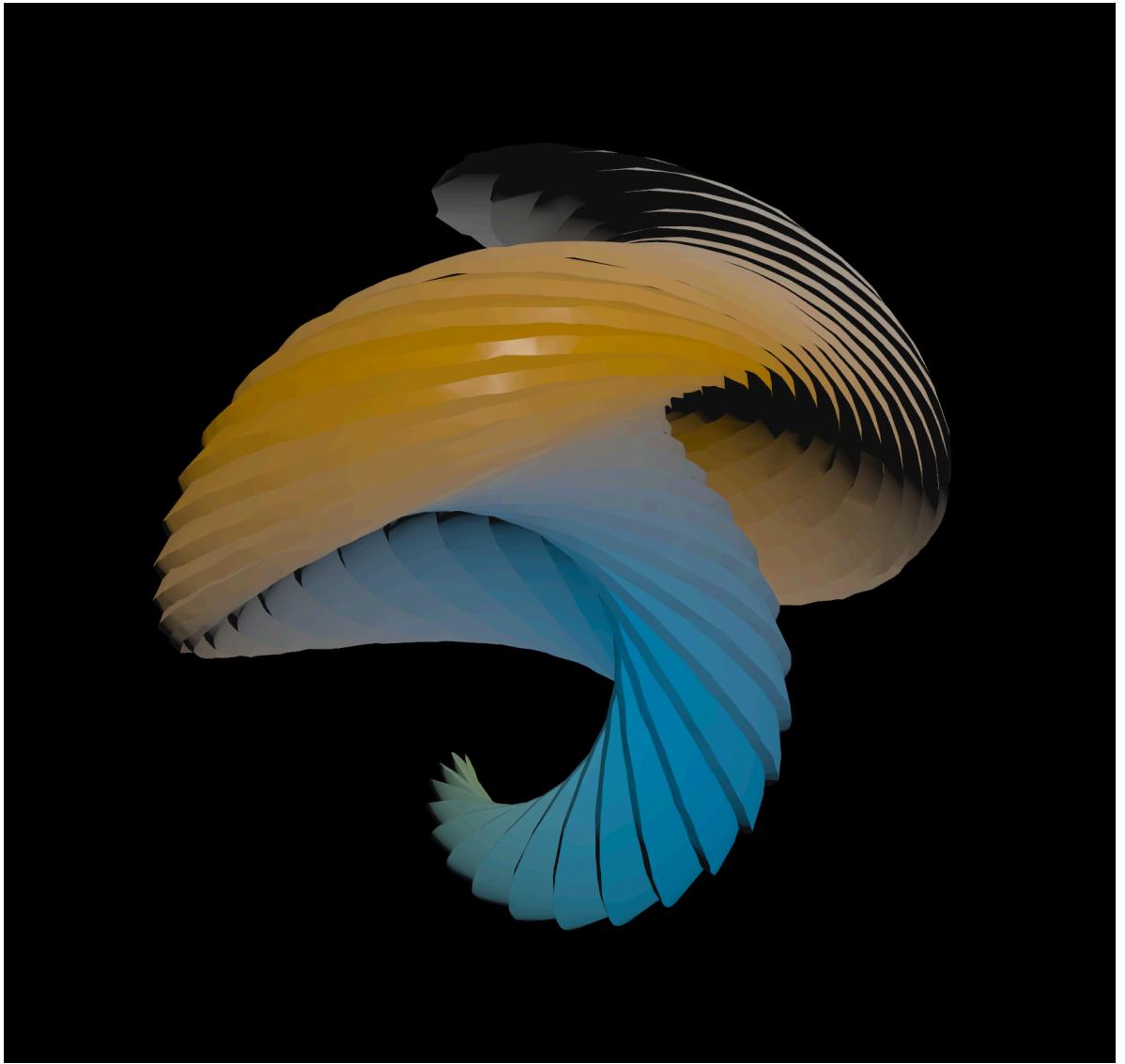
- Number of slices / スライスの数
- Rotational slice angle / 回転スライスの角度



Rotationally slice object.
オブジェクトを回転スライスする。



Slide sliced object.
スライスされたオブジェクトをスライドする。



12-A. Slice cylindrically

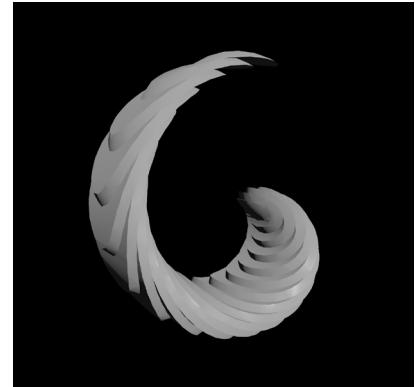
円柱状にカットする



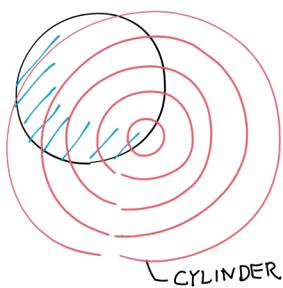
12-rotational_slice-A.hiplc

This example slices sphere cylindrically using For Each loop. The basic process here is to create a scaled cylinder and use it inside a loop with Boolean SOP to slice the sphere. Also at each iteration of the loop the sliced sphere gradually rotates based on a constant rotational axis.

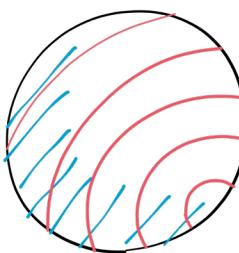
この例では、For Eachループを使って球体を円柱状にスライスしています。基本的なプロセスとしては、スケールした円柱をFor Eachループの中で利用し、球体をBoolean SOPを使ってスライスします。その上で、各ループ処理の中で一定の回転軸を利用してスライスした球体を徐々に回転しています。



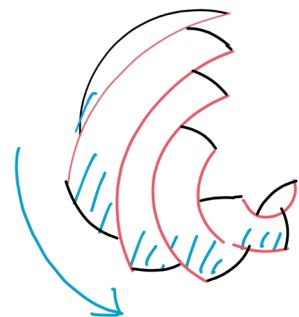
①



②



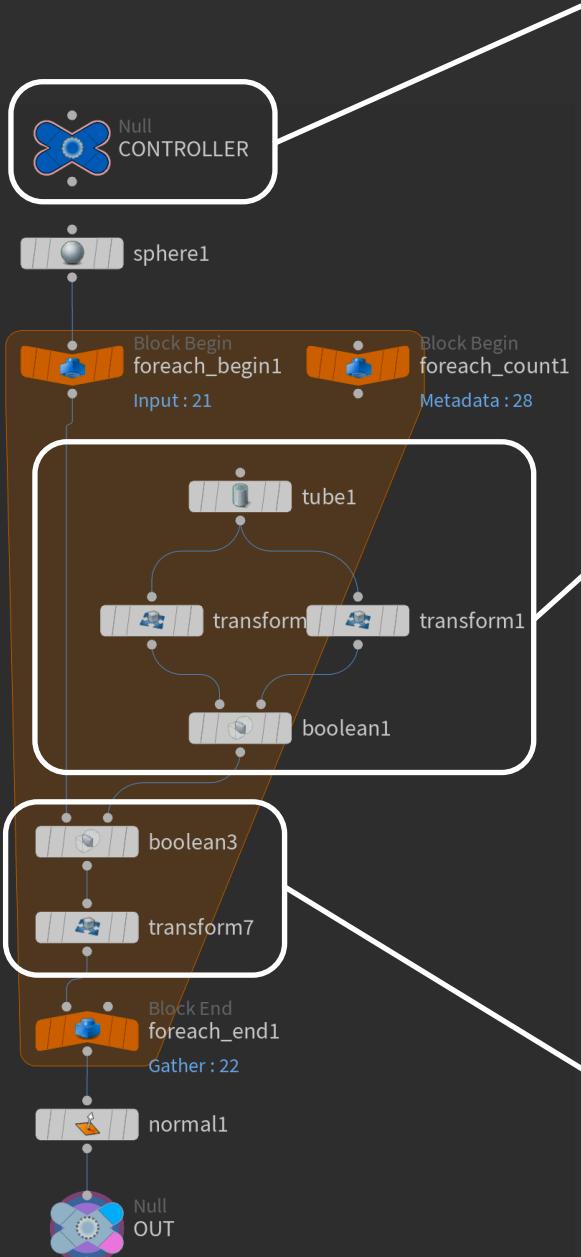
③



Create layers of cylinder.
円柱のレイヤーを作る。

Slice sphere using layers of cylinder with Boolean SOP.
円柱のレイヤーをBoolean SOPで使って球体をスライスする。

Slide sliced object by rotating with one rotational axis.
一軸の回転軸を利用してスライスされたオブジェクトを回転することでスライドする。



Setup parameters

- **size**: size of sphere
- **cut_scale**: scale of cutting tube object
- **split_num**: number of slices
- **angle**: total rotational angle

パラメータの設定

- **size**: 球体のサイズ
- **cut_scale**: 円柱のスケール
- **split_num**: スライスの数
- **angle**: 全体の回転角

Create cutting object from tube

For each iteration of the loop create a cutting object from a tube geometry.

ターゲットの配置

ループ処理ごとにスライス用のジオメトリを円柱から作ります。

Use Boolean SOP to slice

Use Boolean SOP to slice the sphere and rotate the object using Transform SOP.

Boolean SOPを使ってスライスする

Boolean SOPを使って球体をスライスし、Transform SOPを使って回転させます。

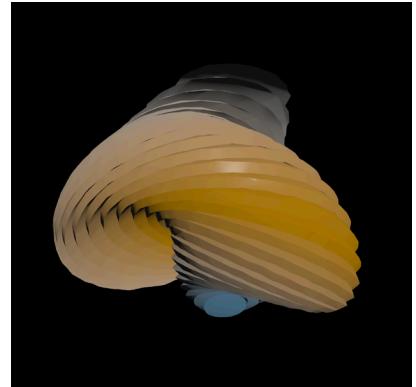
12-B. Slice spherically 球状にカットする



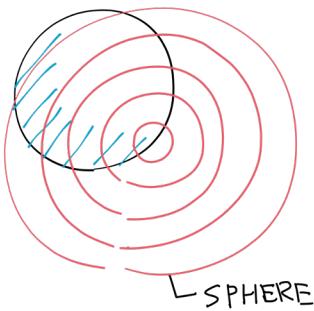
12-rotational_slice-B.hiplc

This example slices sphere spherically using For Each loop. Because this example uses sphere instead of cylinder to slice the object the rotational axis is free compared to cylindrical rotation which makes it a bit more complex to setup.

この例では、For Eachループを使って球体を球状にスライスしています。この例では円柱の代わりに球体を使ってスライスしているため、一軸の回転軸しかない円柱と比べて自由な回転軸を設定することが可能で。そのため、円柱の例と比べるとセットアップは多少複雑になっています。

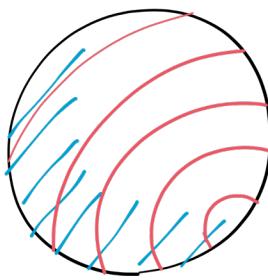


①



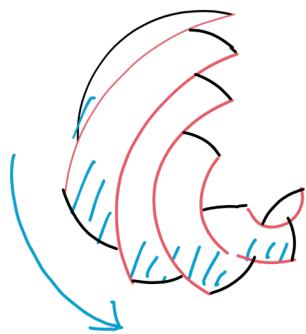
Create layers of sphere.
球体のレイヤーを作る。

②

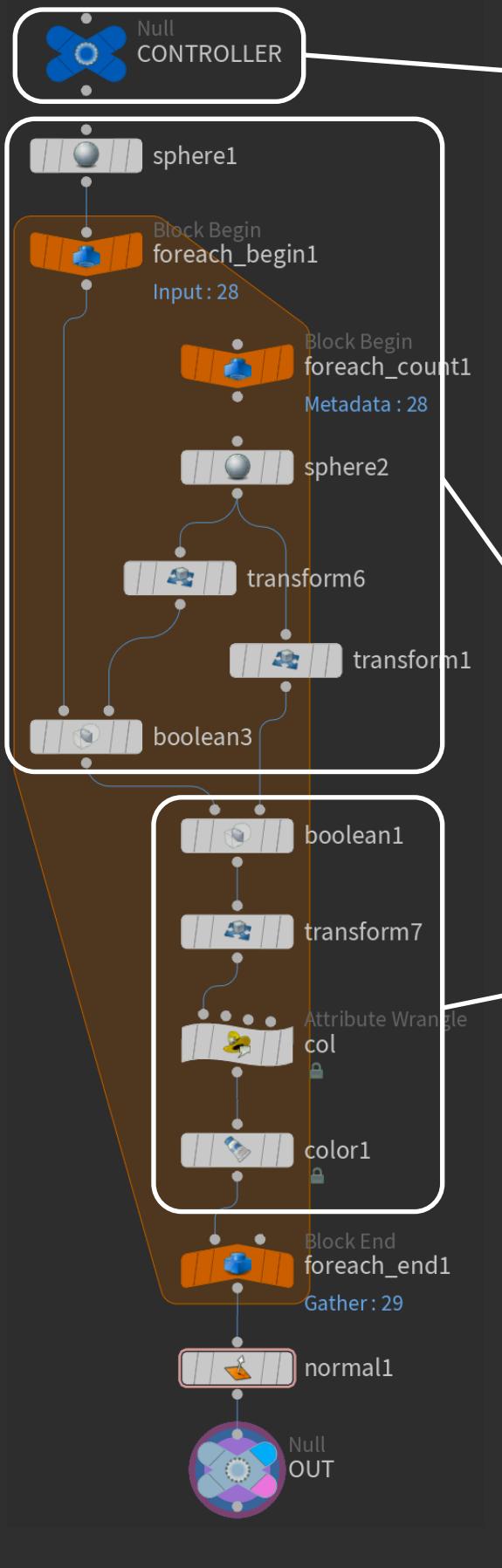


Slice sphere using layers of sphere with Boolean SOP.
球体のレイヤーをBoolean SOPで使って球体をスライスする。

③



Slide sliced object by rotating with three rotational axis.
三軸の回転軸を利用してスライスされたオブジェクトを回転することでスライドする。



Setup parameters

- **size**: size of sphere
- **cut_scale**: scale of cutting tube object
- **split_num**: number of slices
- **angle**: total rotational angle

パラメータの設定

- **size**: 球体のサイズ
- **cut_scale**: 円柱のスケール
- **split_num**: スライスの数
- **angle**: 全体の回転角

Create cutting object from sphere

For each iteration of the loop create a cutting object from a sphere geometry.

ターゲットの配置

ループ処理ごとにスライス用のジオメトリを球体から作ります。

Use Boolean SOP to slice

Use Boolean SOP to slice the sphere and rotate the object with three axis using Transform SOP.

Boolean SOPを使ってスライスする

Boolean SOPを使って球体をスライスし、Transform SOPを使って三軸を利用して回転させます。

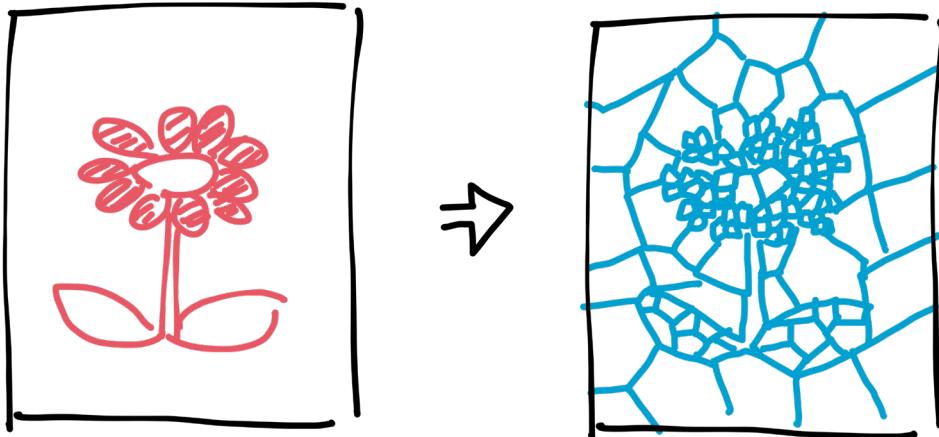
13. Image remeshing 画像を使ってリメッシュする

Use image to remesh the plane. In particular control the shape, size and color of each remeshed cell based on the contrast of an image.

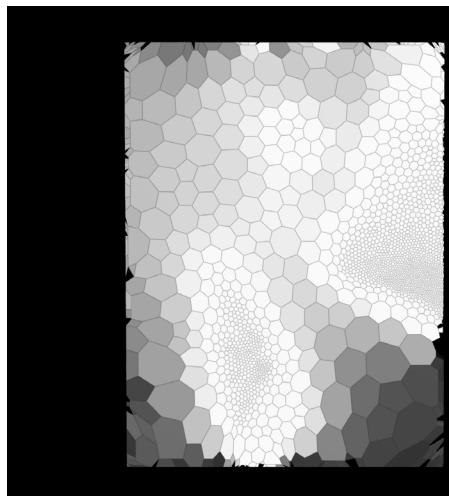
画像を使って平面をリメッシュしてください。具体的には、画像のコントラスト値を使ってリメッシュのセルの形や大きさや色をコントロールしてください。

Preferred parameters / 推奨パラメータ:

- Remesh resolution / リメッシュの解像度
- Minimum and maximum cell size / 最小最大のセルのサイズ



Remesh plane using image.
画像を利用して平面をリメッシュする。



13-A. Remeshing with voronoi

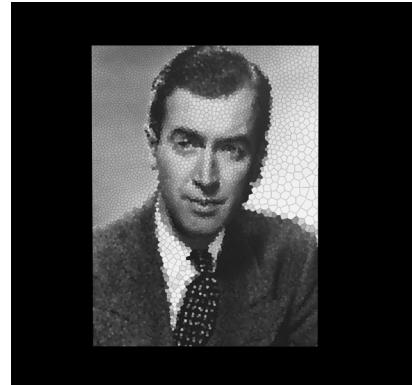
ボロノイを使ってリメッシュする



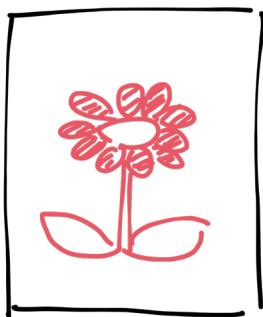
13-image_remeshing-A.hiplc

This example uses image to remesh the plane into colored and sized voronoi cells. Specifically image can be loaded as a grid plane with points that could have color attribute using Attribute from Map SOP. In this case greyscale value is loaded and used it as a density value for Scatter SOP so that points will be scattered on a image plane based on it's contrast. The points then used with Voronoi Fracture SOP to create voronoi cells.

この例では画像を使って平面をサイズと色のついたボロノイセルにリメッシュします。具体的には、Attribute from Map SOPを利用して各点に色の情報がのった平面グリッドを取得することができます。今回の場合は色情報をグレースケールで取得し、それをScatter SOPのdensityの値として使えるように設定します。そうすることで色の明暗に応じて点の密度をコントロールすることができます。その上で、そのポイントからVoronoi Fractureを使ってボロノイセルを作ります。

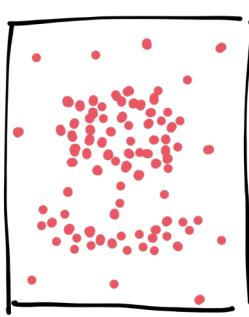


①



Load image as a plane.
画像を平面として読み込む。

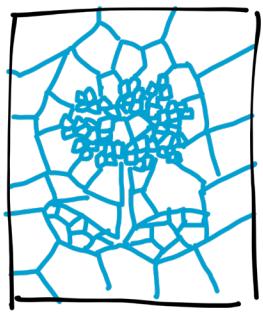
②



Scatter points based on an image greyscale value using Scatter SOP.

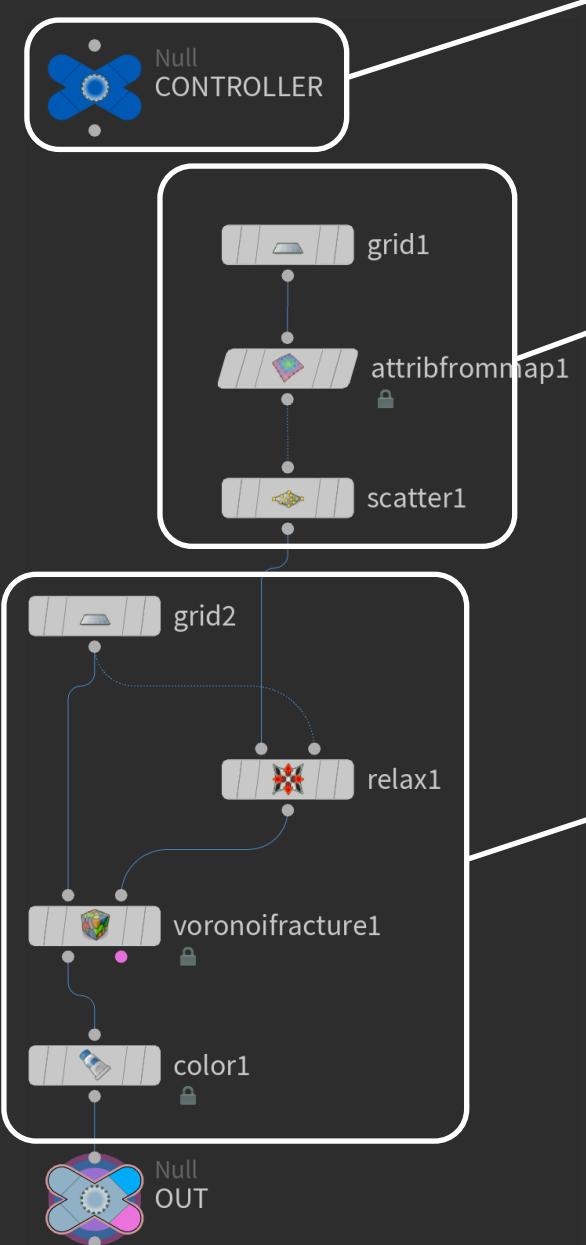
画像のグレースケールの値に応じてScatter SOPを使ってポイントを配置する。

③



Use Voronoi Fracture SOP to create voronoi cells from points.

Voronoi Fracture SOPを使ってポイントからボロノイのセルを作る。



Setup parameters

- **cell_num:** number of voronoi cells
- **smoothness:** smoothness of voronoi

パラメータの設定

- **cell_num:** ボロノイセルの数
- **smoothness:** ボロノイの滑らかさ

Scatter points based on image

Load an image using Attribute from Map SOP and place points based on an image contrast using Scatter SOP.

画像に応じてポイントを配置する

Attribute from Map SOPを利用して画像をロードし、画像のコントラスト値を利用してScatter SOPでポイントを面の上に配置します。

Create voronoi cells

Remesh plane using Voronoi Fracture with points created from image.

ボロノイセルを作る

画像から得られたポイントを利用して、Voronoi Fractureを使って平面をリメッシュします。

13-B. Remeshing with dual mesh

デュアルメッシュでリメッシュする



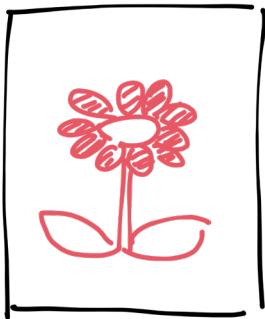
13-image_remeshing-B.hiplc

This example uses dual mesh to remesh the plane. In order to create a dual mesh this example starts by remeshing the plane using Remesh SOP based on greyscale attribute loaded from Attribute from Map SOP. Then use Divide SOP to create a dual mesh.

この例では平面をデュアルメッシュを利用してリメッシュします。デュアルメッシュを作るためには、まずAttribute from Mapで読み込んだグレイスケールの値を利用して、Remesh SOPへ平面をリメッシュします。その上でDivide SOPを利用してデュアルメッシュを作ります。

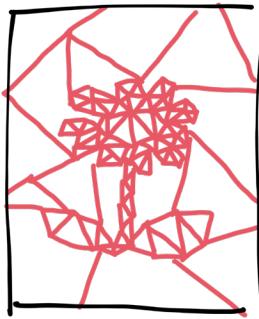


①



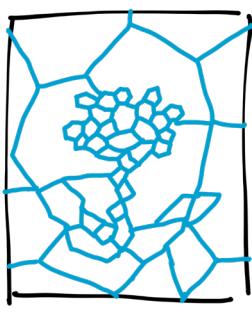
Load image as a plane with greyscale value as an attribute.
画像をアトリビュートとしてグレースケールの値を含んだ平面として読み込む。

②

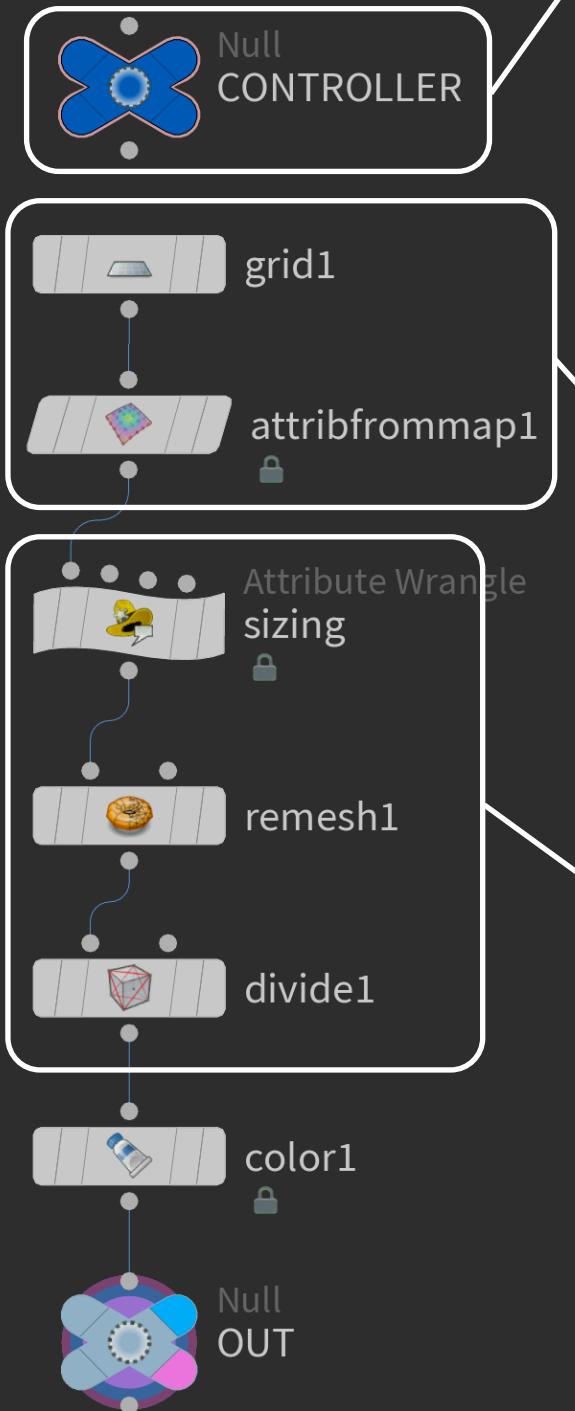


Use Remesh SOP to triangulate the plane using greyscale attribute.
Remesh SOPとグレースケールのアトリビュートの値を使って平面を三角分割する。

③



Use Divide SOP to create dual mesh from triangulated mesh.
Divide SOPを使って三角分割されたメッシュからデュアルメッシュを作る。



Setup parameters

- **min_size**: minimum mesh size
- **max_size**: maximum mesh size

パラメータの設定

- **min_size**: メッシュの最小サイズ
- **max_size**: メッシュの最大サイズ

Load image

Load an image using Attribute from Map SOP as a greyscale attribute on a plane.

画像をロードする

Attribute from Map SOPを利用して画像をグレースケールのアトリビュートとしてロードします。

Create dual mesh

Use Remesh SOP to remesh plane using greyscale attribute from an image plane then use Divide SOP to create dual mesh.

デュアルメッシュを作る

Remesh SOPを使って、画像の平面を拡張点のグレースケール値を元にリメッシュします。そこからDivide SOPを利用してデュアルメッシュを作ります。

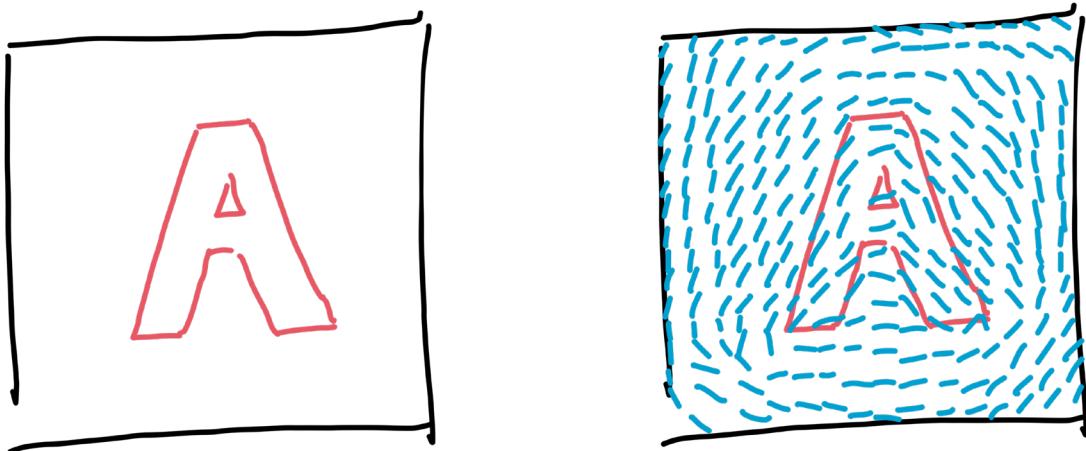
14. Vector field control オブジェクトでベクトル場を操作する

Visualize a vector field affected by a text. In particular use text outline as a driving force to control the direction of vector field. Create a vector field in either 2d or 3d.

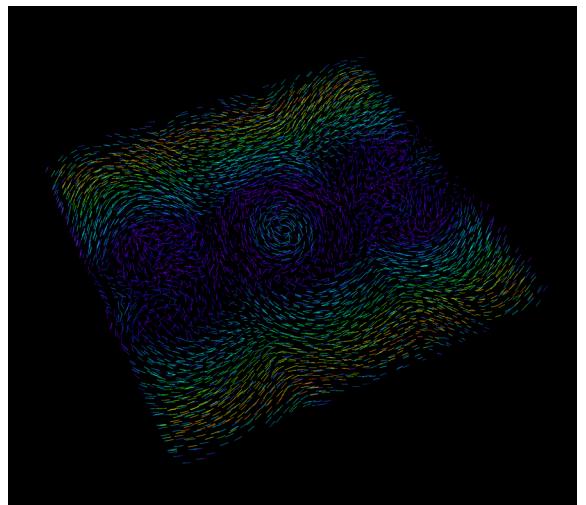
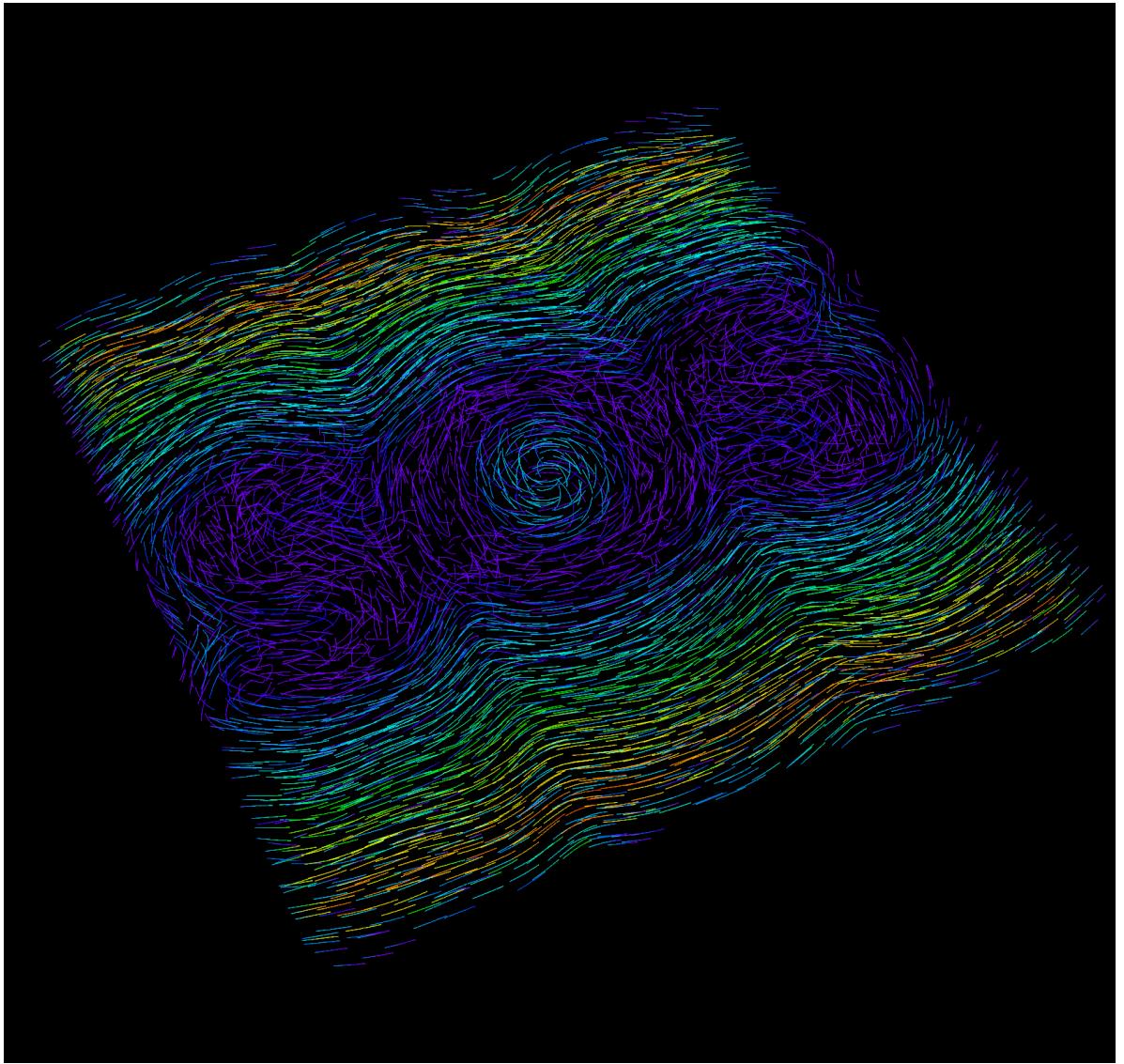
テキストによって影響を受けるベクトル場を作ってください。具体的には、テキストのアウトラインがベクトル場の方向をコントロールするようにしてください。その際、二次元か3次元のベクトル場を作ってください。

Preferred parameters / 推奨パラメータ:

- Text to control vector field / ベクトル場をコントロールするテキスト
- Text force strength / テキストによる力の強さ



Visualize a vector field affected by a text geometry.
テキストのジオメトリによって影響を受けたベクトル場を作る。



14-A. 2D vector field

二次元のベクトル場



14-vector_field-A.hipsc

This example creates a 2d vector field affected by text outline. Text outline's tangent vector will determine the direction of the nearby vector field.

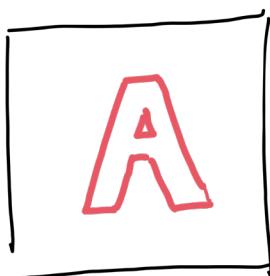
Since vector field itself is a collection of vectors so you cannot really see them unless you specifically visualize it in some way. This example visualize the 2d vector field by using scattered points on a grid plane as a starting point of a line which is rotated based on the nearby tangent vector of the text outline.



この例ではテキストのアウトラインに影響を受けた二次元のベクトル場を作ります。テキストのアウトラインの接線ベクトルによってベクトル場のベクトルの方向が決定されます。

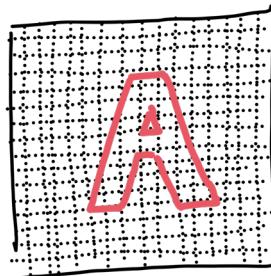
ベクトル場自体は、目に見えないベクトルの集合であるため、意図的に可視化しない限りは見ることができません。この例では、それを平面にラインの始点となる無数のポイントを配置し、ポイントの近くのテキストアウトラインの接線方向を利用してラインを回転することで二次元のベクトル場を表現しています。

①



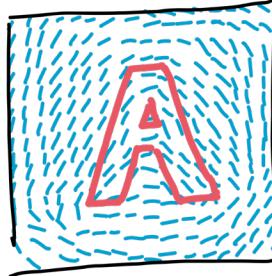
Create a plane as a boundary and a text geometry.
バウンダリとしての平面とテキストのジオメトリを作る。

②

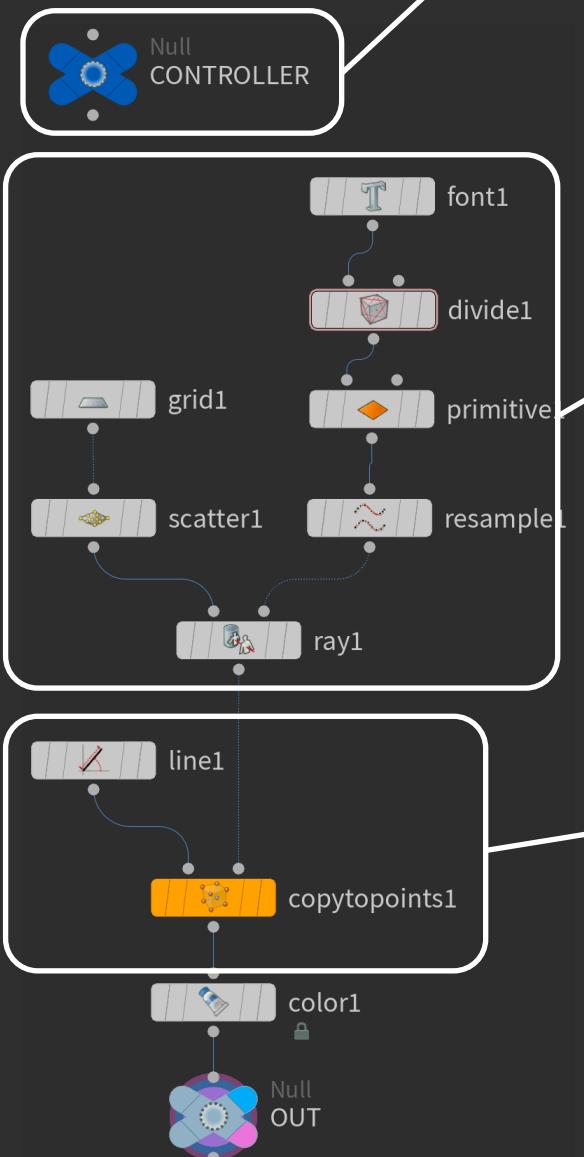


Get a direction from each point on grid plane to text geometry outline.
グリッド平面上の各ポイントからテキストジオメトリのアウトラインへの方向を取得する。

③



Use retrieved direction to create vector field line for each point.
各ポイントにベクトル場としてのラインを事前に取得した方向を利用して作る。



Setup parameters

- **pt_num:** number of vectors
- **length:** length of each vector

パラメータの設定

- **pt_num:** ベクトル場のベクトルの数
- **length:** 各ベクトルの長さ

Create vector field base points

Create base points on a grid plane which inherits the nearby tangent vector of text outline as a normal vector.

ベクトル場のベースのポイントを作る

平面の上にベクトル場のベースとなる複数のポイントを作り、近くのテキストのアウトラインの接線方向を法線として継承します。

Visualize vector field with lines

Visualize 2d vector field by copying line to each point automatically rotated with it's normal attribute.

ラインでベクトル場を可視化する

二次元のベクトル場を、ラインをポイントの位置にコピーすることで可視化します。ラインはポイントの法線ベクトルを利用して自動的に回転します。

14-B. 3D vector field

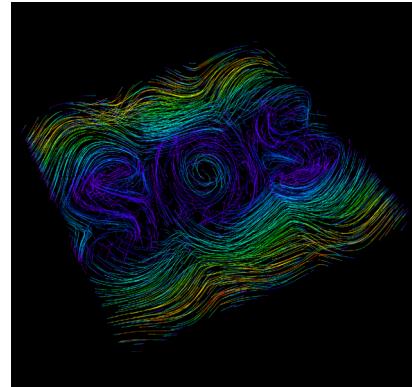
三次元のベクトル場



14-vector_field-B.hiplc

In lot of cases you might want to make 3d vector field instead of 2d vector field. In this case one of the easiest way to create 3d vector field is to use velocity volume.

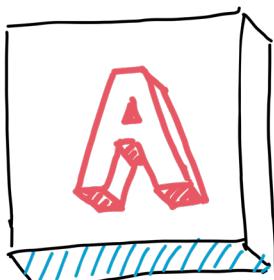
In order to modify a velocity volume using text geometry it is a good idea to use Volume Wrangle to write VEX to control the vector directions since this is one of the most flexible way. You can visualize the flow of the vector field made with velocity volume by using Volume Trail SOP.



多くの場合、二次元のベクトル場ではなく三次元のベクトル場を作りたい場合もあるかと思います。このような場合、三次元のベクトル場を作る最も簡単な方法の一つとして考えられるのは、三次元のベロシティのボリュームを利用することです。

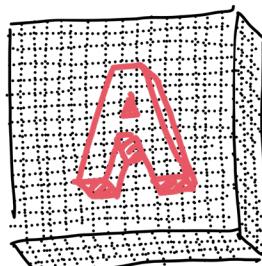
このベロシティのボリュームが空間に置かれたテキストによって影響を受けるようにするために、Volume Wrangleを使ってVEXを書いてベクトル操作をするのが最もフレキシブルな方法かと考えられます。最終的に、Volume Trail SOPを利用してベロシティのボリュームを可視化することができます。

①



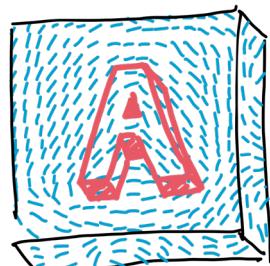
Create a volume as a boundary and a text geometry.
バウンダリとしてのボリュームとテキストのジオメトリを作る。

②

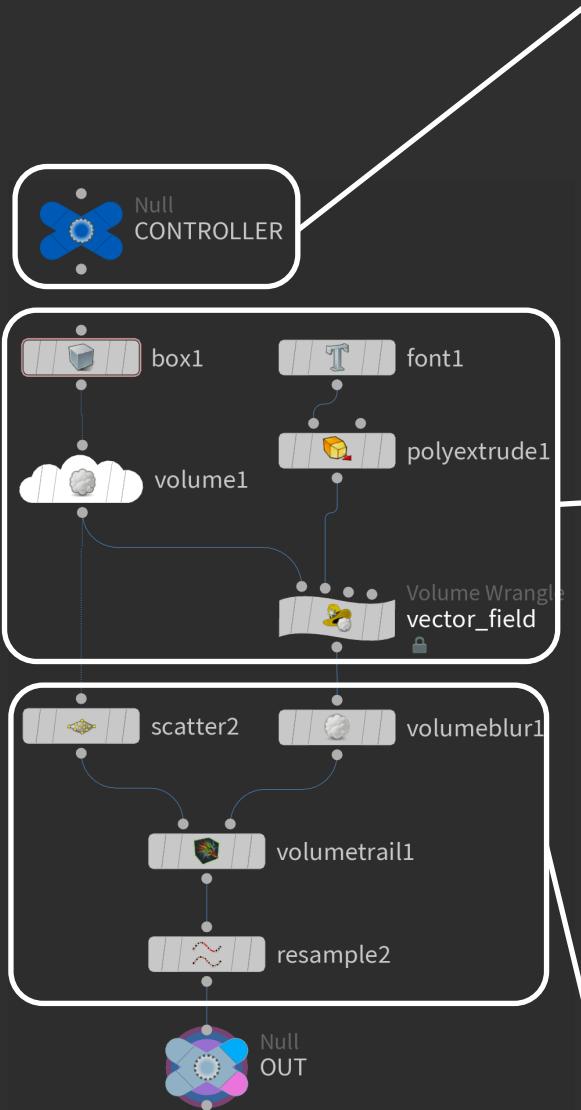


Get a direction from each voxel on volume to text geometry and set the volume value with it.
ボリュームの各ボクセルからテキストジオメトリへの方向を取得し、ボリュームの値として設定する。

③



Use volume value together with VolumeTrail SOP to visualize 3d vector field.
ボリュームの値とVolumeTrail SOPを利用して三次元のベクトル場を可視化する。



Setup parameters

- **pt_num:** number of vectors
- **length:** vector length
- **blur:** blurring value for vector field
- **y smoothness:** noise smoothness for y axis
- **y amount:** noise amount for y axis

パラメータの設定

- **pt_num:** ベクトル場のベクトルの数
- **length:** ベクトルの長さ
- **blur:** ベクトル場にかけるぼかしの量
- **y smoothness:** Y軸のノイズの滑らかさ
- **y amount:** Y軸のノイズの大きさ

Create 3d volume vector field

Create and modify a 3d vector field using Volume Wrangle SOP based on the extruded text geometry.

三次元のボリュームのベクトル場を作る

三次元のベクトル場をVolume Wrangle SOPを利用して作ります。その際、押し出したテキストジオメトリをもとにそのベクトル方向を決定します。

Visualize vector field

Use Volume Trail SOP to visualize the 3d vector field with specific number of points inside the volume region.

ベクトル場を可視化する

ボリュームの領域の中に指定の数のポイントを配置し、Volume Trail SOPを利用して三次元のベクトル場を可視化します。

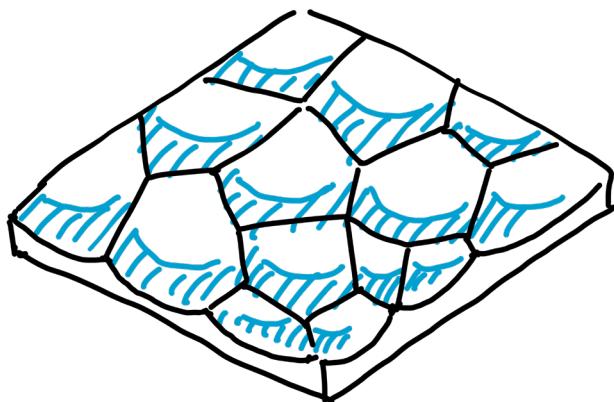
15. **Gouged surface** サーフェスを削る

Gouge any input object to create a chiseled pattern.

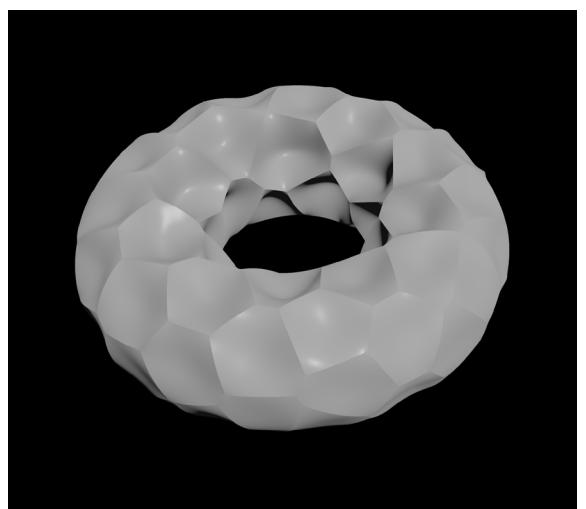
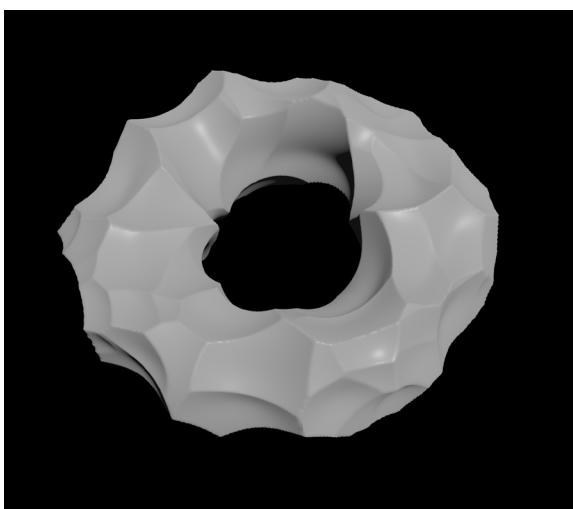
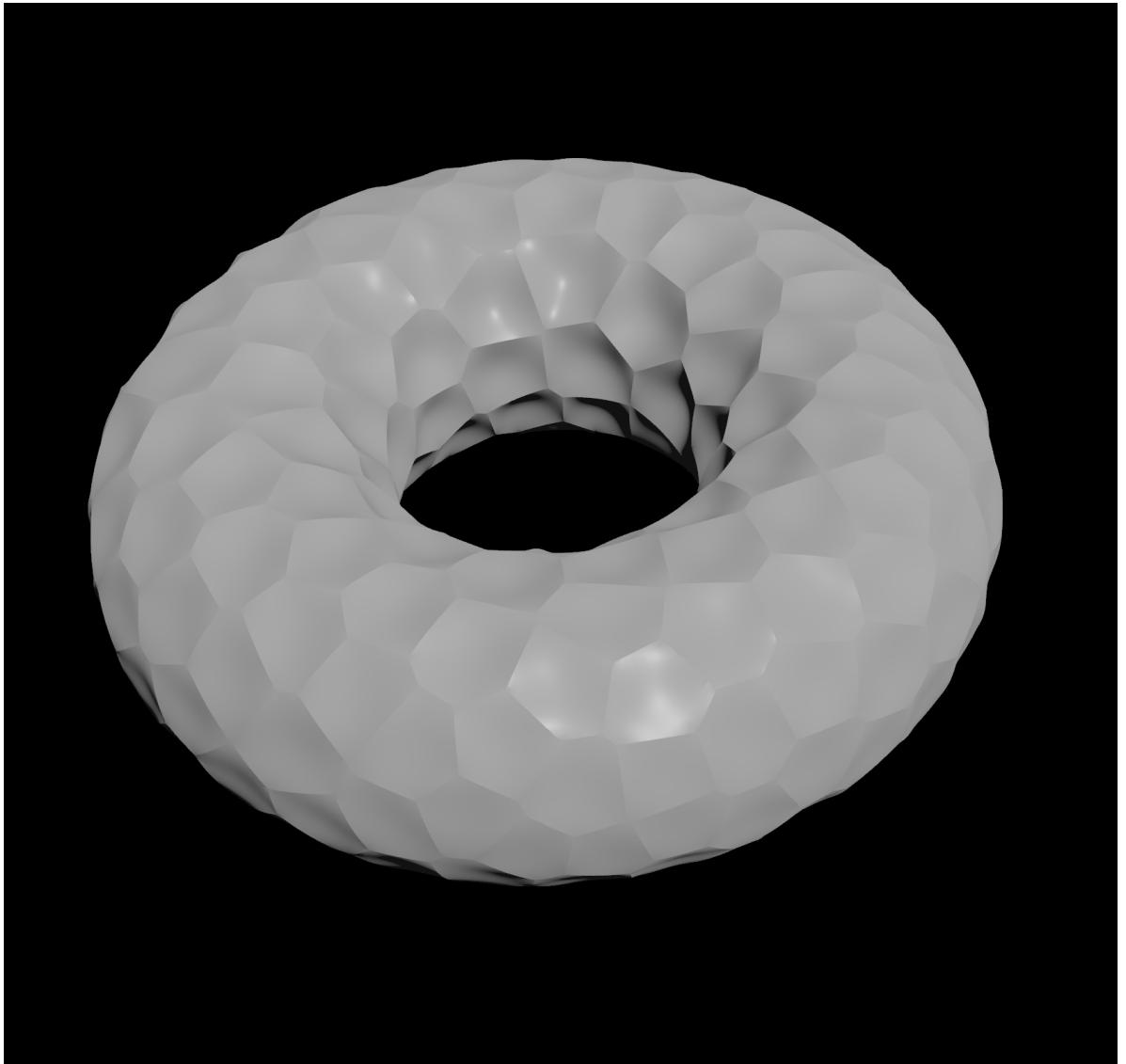
任意のジオメトリに彫刻刀で彫ったような模様をつけてください。

Preferred parameters / 推奨パラメータ:

- Chiseled pattern resolution / 彫り模様の解像度
- Chiseled depth / 彫りの深さ



Create gouged pattern on any input geometry.
任意のジオメトリに彫り模様を作る。



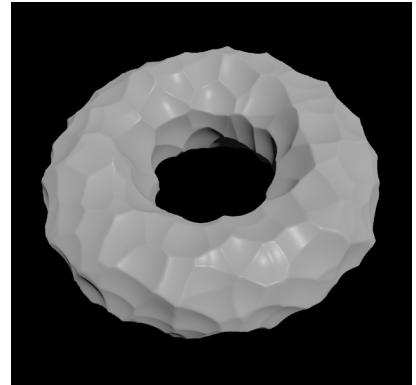
15-A. Gouge using Mountain

Mountainを使って削る

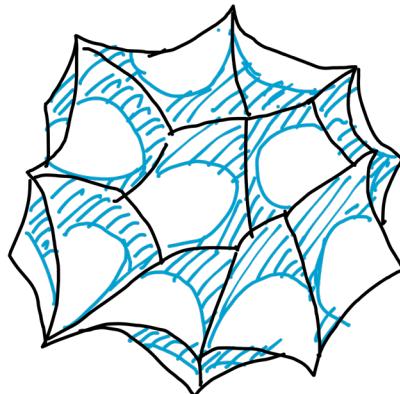


15-gouged_rock-A.hiplc

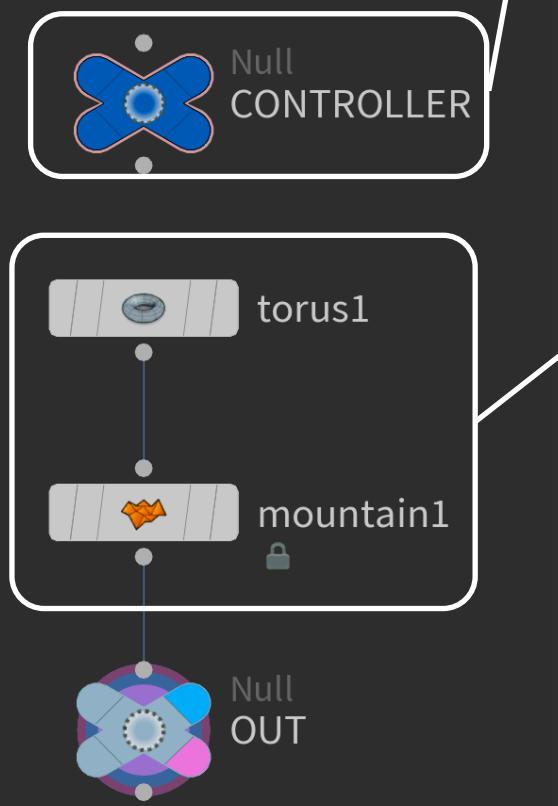
There are few ways to create a gouged surface but the simplest way I could think of is to use Mountain SOP as it is by just changing some parameters and nothing else. There is a noise type called Worley which is often used for wavy ocean but by tweaking the parameter it could easily be used as a gouged surface pattern. The downside of using this method is that you cannot make the height of the ridge the same which breaks the original shape when you made the amount of noise too big.



いくつか彫り模様のサーフェスを作る方法が考えられますが、一番簡単なのはMountain SOPをパラメータを変えるだけでそのまま使ってしまう方法です。特に、Mountain SOPのNoise Typeというパラメータの中にWorleyというものがあり、普段は海の波を表現する際によく使われるのですが、サーフェスを彫った模様を作る際にそのまま利用することができます。難点があるとすると、リッジ線の高さを一定に保つことができないため、強くノイズをかけると元の形状が崩れやすいという点です。



Use Mountain SOP to create gouged pattern on a geometry.
Mountain SOPを使ってジオメトリの上に彫り模様を作る。



Setup parameters

- **height:** height of gouge pattern
- **element size:** size of gouge pattern

パラメータの設定

- **height:** 彫りの高さ
- **element size:** 彫りのサイズ

Create gouged surface

Use Mountain SOP with Worley noise type to create gouged like pattern on an input surface.

彫り模様のサーフェスを作る

Mountain SOPでWorleyというノイズのタイプを利用して、入力したサーフェス上に彫り模様を作ります。

15-B. Gouge using voronoi

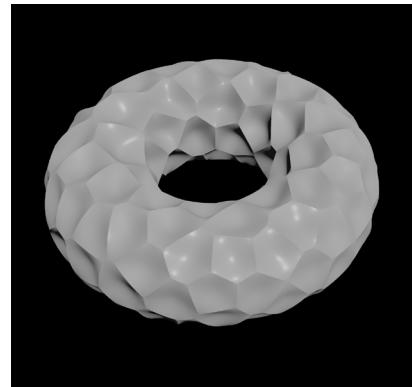
ボロノイを使って削る



15-gouged_rock-B.hiplc

When you want more control in order to create gouged surface especially preserving the height of the ridge edges then the process will be a bit more complex than using one Mountain SOP which could be used for any input geometry.

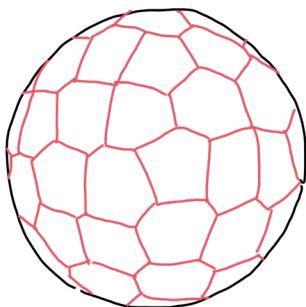
In this example Voronoi Fracture is used to create gouge pattern base and using VEX to create a smooth dent for each voronoi cell. The amount of dent is calculated based on the shortest distance between each cell's point and cell's outline curve.



もっとコントロールが効く彫り模様を作りたい場合、特にリッジ線の高さを元のジオメトリのシェイプに合わせて作りたい場合、Mountain SOPよりも多少複雑なプロセスを踏まないといけなくなりますが、どのようなジオメトリにも適用できます。

この例では、Voronoi Fracture SOPを使って彫り模様のベースを作り、個々のボロノイのセルに対してVEXを使ってなめらかな凹みを作っています。凹みの深さはセルの中のポイントとセルのアウトラインとの最短距離をベースに計算しています。

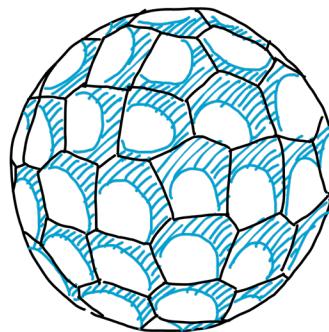
①



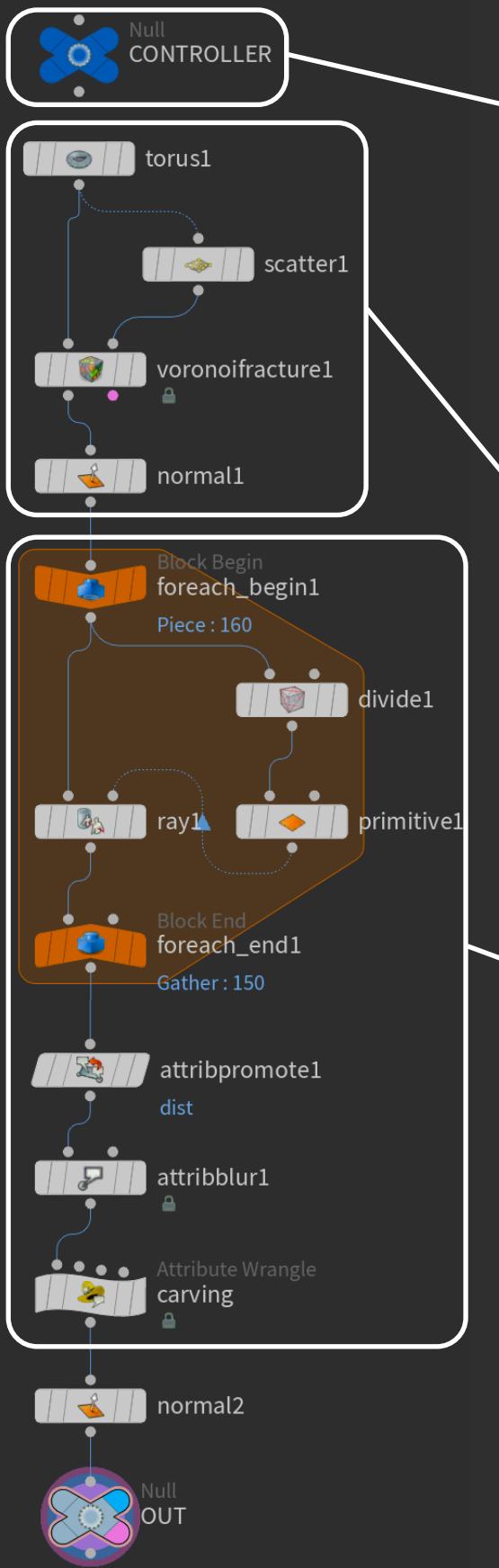
Create voronoi cells on any input geometry using Voronoi Fracture SOP.

任意のジオメトリからVoronoi Fracture SOPを使ってボロノイのセルを作る

②



Create dent for each voronoi cell.
各ボロノイセルに凹みをつける。



Setup parameters

- **num_cell:** number of voronoi cells
- **height:** height of each cell
- **smoothness:** smoothness of each cell

パラメータの設定

- **num_cell:** ボロノイセルの数
- **height:** セルの高さ
- **smoothness:** セルの滑らかさ

Create voronoi cells

Create voronoi cells from input geometry using Voronoi Fracture SOP.

ボロノイセルを作る

Voronoi Fracture SOPを使って任意のジオメトリからボロノイセルを作る。

Create a dent for each cell

Create a dent for each voronoi cell using VEX by using a shortest distance between points in each cell and an outline of the cell.

個々のセルに凹みを作る

VEXを利用して個々のボロノイのセルに凹みをつけます。凹みの量はセルの中のポイントからセルのアウトラインへの最短距離の情報を用いて作ります。

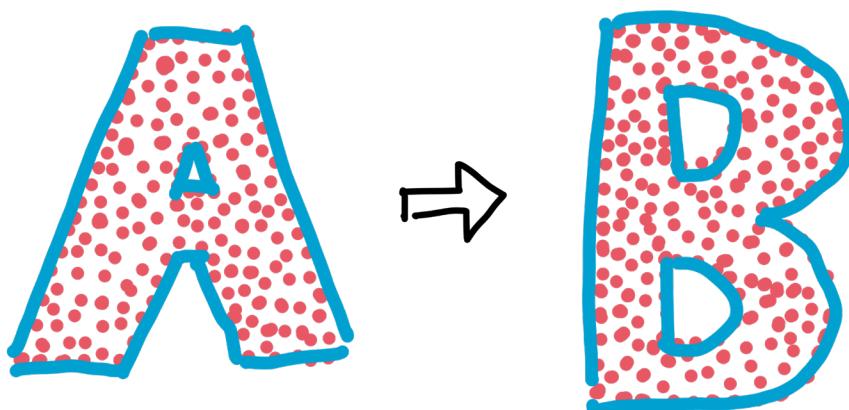
16. Particle morphing パーティクルをモーフィングする

Create a particle morphing so that one shape made with collection of particles (points) morphs to another shape by time or slider value. If possible try to add a spreading effect when morphing from one shape to another.

パーティクル(点)の集合でできた一つの形から、別の形へモーフィングするエフェクトを作ってください。その際時間やスライダーの値を変えることでモーフィングできるようにしてください。可能であれば、一つの形から別の形へモーフィングする際、パーティクルが拡散するエフェクトを加えてください。

Preferred parameters / 推奨パラメータ:

- Morphing time / モーフィングの時間
- Source and target shape / スタートとゴールの形
- Number of particles / パーティクルの数



Create particle morphing by having a source and target particles.
ソースとターゲットのパーティクルを用意してパーティクル間のモーフィングを作る



ORANGE



SMOOTHIE



JELLIES

16-A. Linear morphing

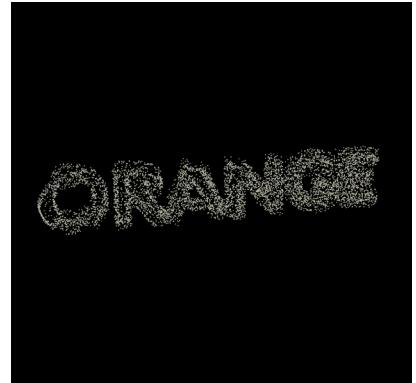
線形モーフィング



16-particle_morphing-A.hiplc

In order to create a particle morphing you have to have a same number of particles on a source shape and a target shape. Keep that in mind the rest is pretty easy when you just want to make a linear particle morphing.

Specifically you can just use Blend Shape SOP in order to create a morphed shape from source shape particles to target particles. A tip here is to sort source and target points using same rule since the morphing is done between points with same id. In this example points are sorted using Sort SOP by X axis.



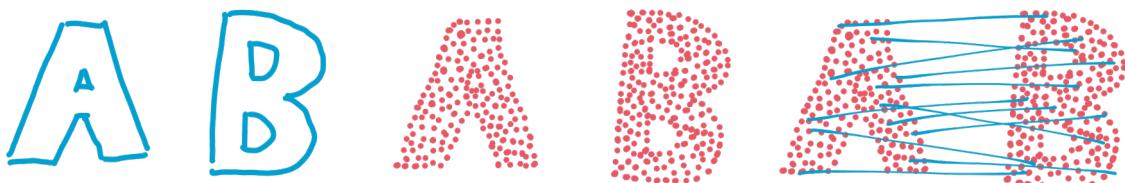
パーティクルモーフィングを作るには、まず前提としてモーフ前のパーティクルとモーフ後のパーティクルの数が揃っている必要があります。それさえ守れば、直線的なパーティクルのモーフィングは非常に簡単に作ることができます。

具体的にはBlend Shape SOPを使うことで直線的なパーティクルモーフィングを作ることができます。ティップスとしては、モーフ元とモーフ後のパーティクルの順番を同じルールで並び替えしておくのが良いかと思います。というのもモーフィングは同じidをもつポイント間で行われるからです。この例では、Sort SOPを利用してX軸に沿ってポイントを並び替えています。

①

②

③



Prepare a pair of text geometry one for source and another one for target.
テキストのジオメトリをソースとターゲット分ペアとして作る。

Create points as particles on the source and the target text geometry.
ソースとターゲットのテキストジオメトリの上にパーティクルとしてのポイントを作る。

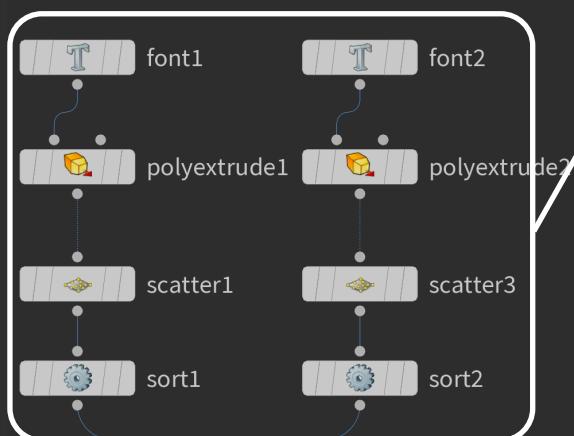
Create a linear morphing between source and target particles using Blend Shape SOP.
Blend Shape SOPを使ってソースとターゲットのパーティクル間で線形のモーフィングを作る。

Setup parameters

- **particle_num:** number of particles
- **interpolate:** time value to morph

パラメータの設定

- **particle_num:** パーティクルの数
- **interpolate:** モーフィングの時間



Set source and target particles

Create source and target particles from a text geometry.

ソースとターゲットのパーティクルを作る
ソースとターゲットのパーティクルをテキストのジオメトリから作ります。

Use Blend Shape SOP to morph

User Blend Shape SOP to create linear morphing between two sets of particles.

Blend Shape SOPを使ってモーフィング

Blend Shape SOPを使って、二種類のパーティクルから線形のモーフィングを作ります。

16-B. Spread morphing 拡散モーフィング



16-particle_morphing-B.hiplc

If you want to have more control over morphing Blend Shape SOP is not good enough but you might want to use VEX for more flexibility.

In this example Wrangle SOP is used instead of Blend Shape SOP in order to morph in order to add a spreading effect during morphing. Specifically noise function together with simple trigonometry is used to add the spreading effect during the morphing.



もしよりモーフィング具合をコントロールしたい場合は、Blend Shape SOPでは役不足感があり、VEXを使ったほうがよりフレキシブルにコントロールが可能かもしれません。

この例では、Blend Shape SOPの代わりにWrangle SOPを利用してモーフィングに拡散のエフェクトを加えています。具体的にはノイズ関数と簡単な三角関数を利用してモーフィングに拡散のエフェクトを加えられるようにしています。



① Prepare a pair of text geometry one for source and another one for target.
テキストのジオメトリをソースとターゲット分ペアとして作る。

② Create points as particles on the source and the target text geometry.
ソースとターゲットのテキストジオメトリの上にパーティクルとしてのポイントを作る。

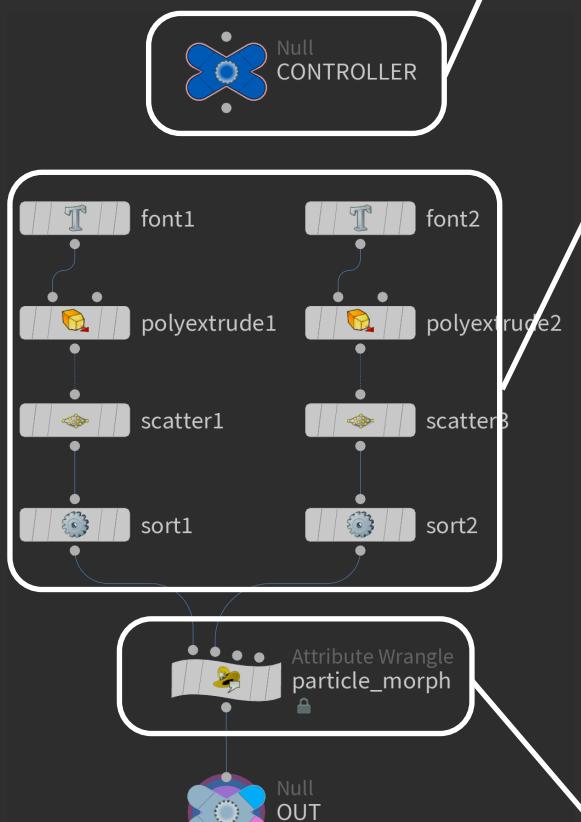
③ Create a spread morphing between source and target particles using VEX.
VEXを使ってソースとターゲットのパーティクル間で拡散しながらモーフィングするようとする。

Setup parameters

- **particle_num:** number of particles
- **interpolate:** time value to morph
- **randomness:** spreading randomness
- **spreadness:** spreading amount

パラメータの設定

- **particle_num:** パーティクルの数
- **interpolate:** モーフィングの時間
- **randomness:** 拡散のランダム具合
- **spreadness:** 拡散量



Set source and target particles

Create source and target particles from a text geometry.

ソースとターゲットのパーティクルを作る
ソースとターゲットのパーティクルをテキストのジオメトリから作ります。

Morphing with spreading effect

Use VEX to create a morphing effect together with spreading effect based on noise function.

拡散エフェクト付きモーフィング

VEXを使って拡散エフェクト付きのモーフィングをノイズ関数をベースに作成します。

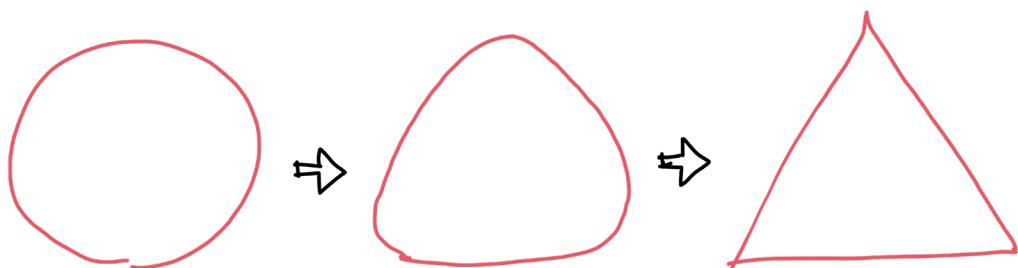
17. **Silhouette morphing** シルエットをモーフィングする

Prepare three geometries and create silhouette outline out of them. Then create a morphing animation between those three silhouette outlines. If possible try to have a ramp parameter to control the easing of morphing animation.

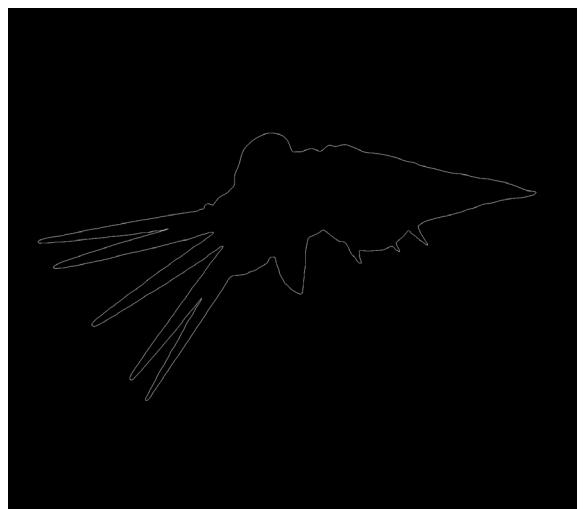
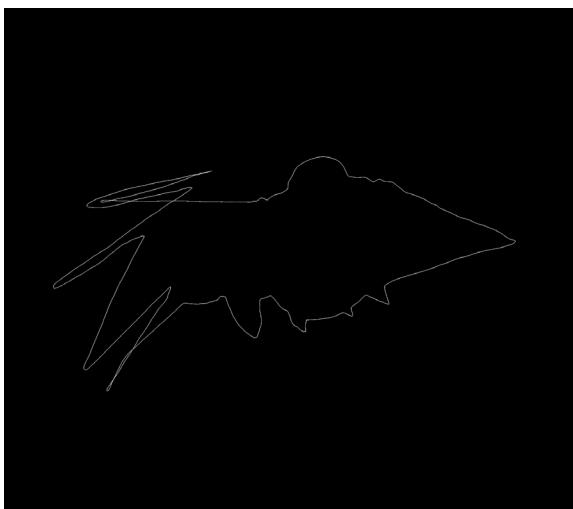
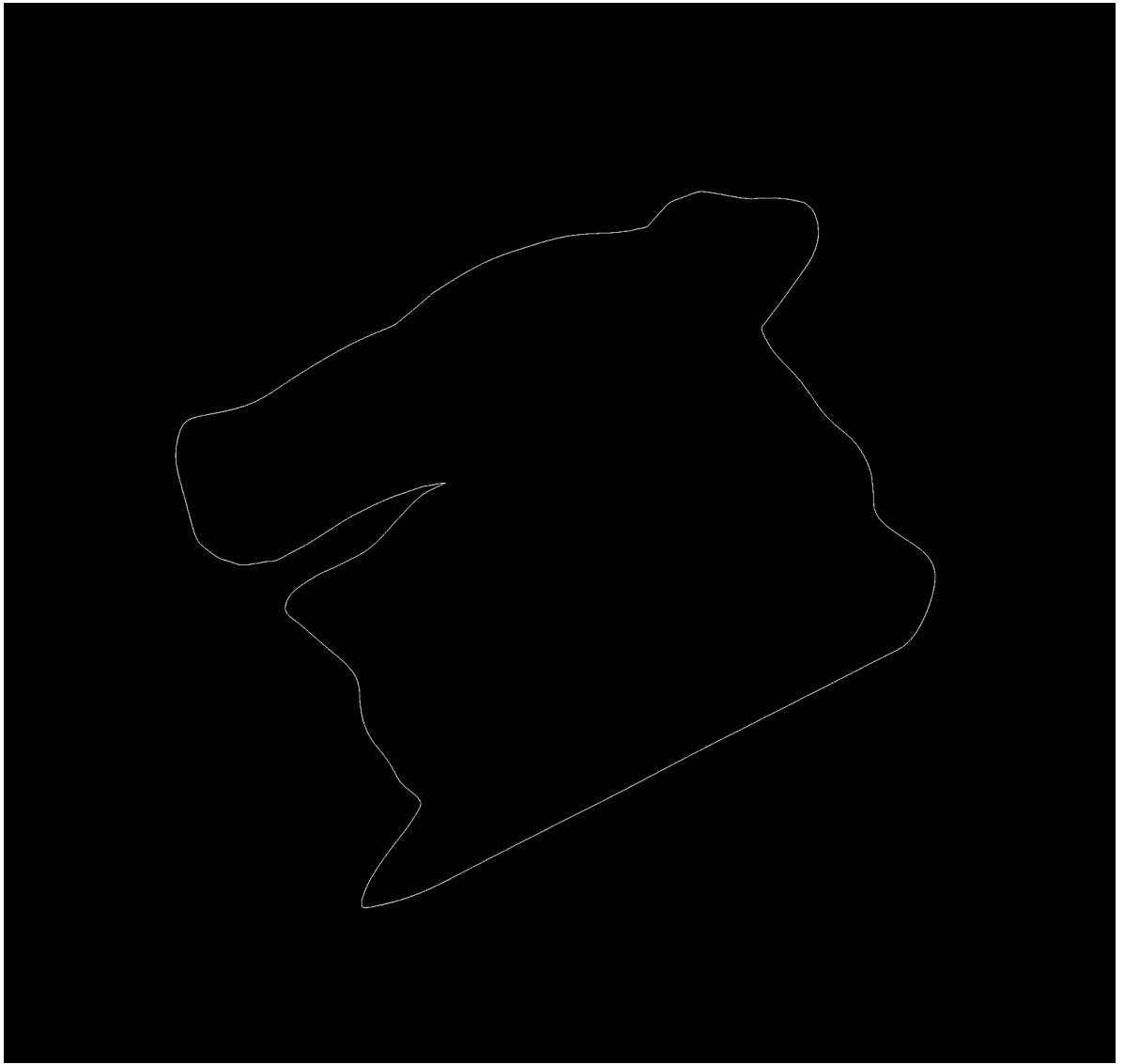
三つのジオメトリを用意して、それらからシルエットを曲線として作ってください。その上で、その三つのアウトラインを行き来するモーフィングのアニメーションを作ってください。そのとき、可能であればイージングのコントロールをランプパラメータを使ってできるようにしてみてください。

Preferred parameters / 推奨パラメータ:

- Morphing time / モーフィング時間
- Easing ramp / イージング設定



Create a morphing between three silhouette outlines.
三個のアウトライン間でモーフィングを作る。



17-A. Linear morphing

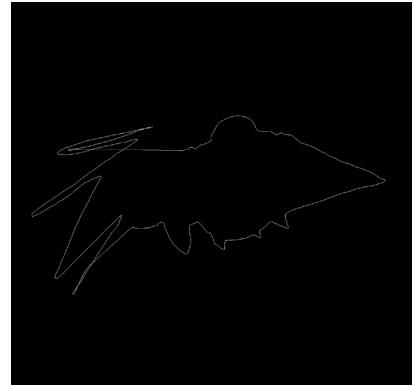
線形モーフィング



17-silhouette_morphing-A.hiplc

The method for silhouette morphing is similar to that of particle morphing but in this example the morphing is done between multiple number of outline shapes. The prerequisite is the same with particle morphing, that is to have same number of points between all morphing shapes.

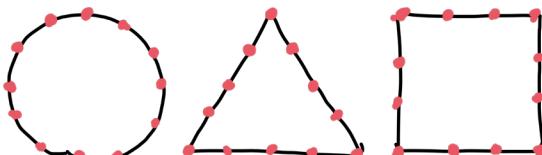
This example is using Blend Shape SOP in order to create linear morphing between shapes. Additionally it uses Delete SOP to pick the corresponding pair of shapes for source and target based on the morphing timing.



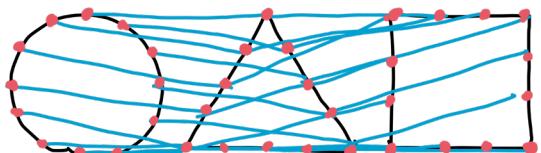
シルエットのモーフィングの手法はパーティクルのモーフィングの手法と似ていますがこの例では複数のシルエット形状の間でモーフィングするような形になっています。前提として、パーティクルモーフィングと同じようにモーフィングする複数の形状間で同じ数のポイントを持っている必要があります。

この例ではBlend Shape SOPを使って、形状間の線形モーフィングを行っています。その際、モーフィングの時間に応じてDelete SOPを使い、モーフ元とモーフ後の形状をペアとして複数の形状から取り出しています。

①

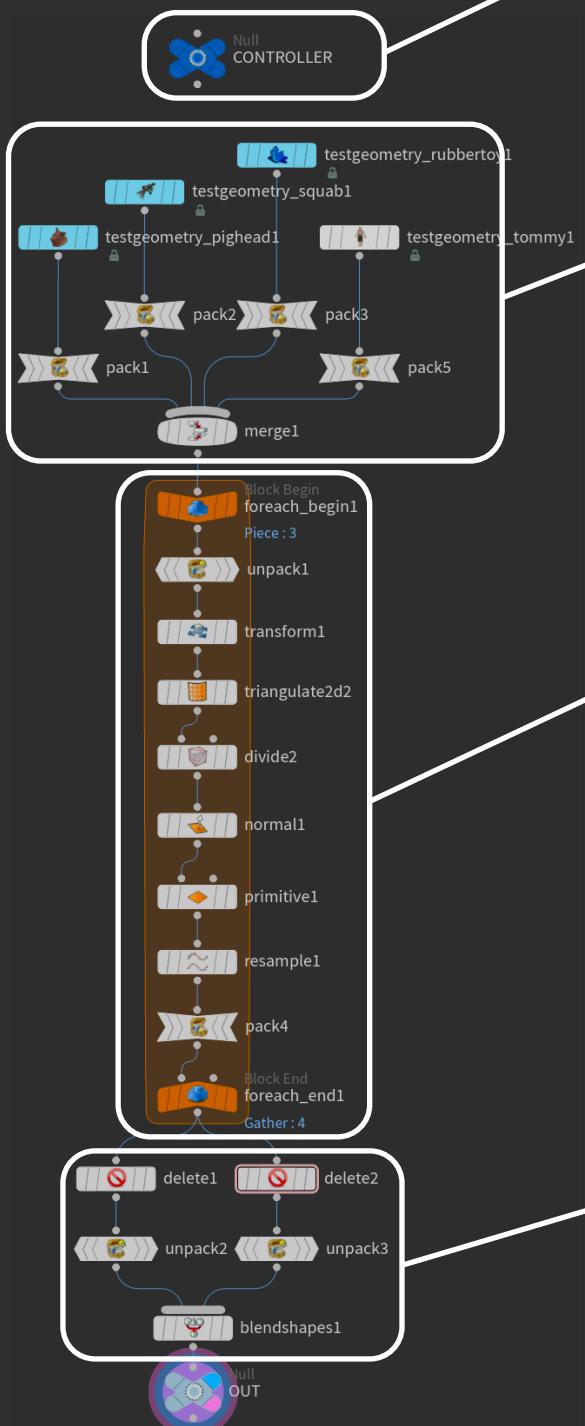


②



Prepare three silhouette outlines with same number of points.
同じ数だけのポイントをもった三つのシルエットのアウトラインを作る。

Create a linear morphing between three silhouette outlines using Blend Shape SOP.
Blend Shape SOPを利用して三つのシルエットのアウトライン間で線形のモーフィングを作る。



Setup parameters

- **tween:** morphing time

パラメータの設定

- **tween:** モーフィングの時間

Prepare multiple shapes

Prepare multiple shapes in order to create silhouette.

複数の形状を用意する

シルエット作成のために複数の形状を用意します。

Create silhouette from a shape

Use For Each loop to create silhouette curve for each shape mainly using Triangulate SOP.

形状からシルエットを作る

For Eachループの中で、形状ごとに主に Triangulate SOPを利用してシルエットの曲線を作ります。

Pick two shapes to morph

Use Delete SOP with morphing time parameter to get two shapes for morphing and use Blend Shape SOP to actually morph linearly.

モーフィングのために二つの形状を得る

Delete SOPとモーフィングの時間のパラメータを利用して、モーフィングに使う二つの形状を取得し、Blend Shape SOPを利用して線形モーフィングを行います。

17-B. Morphing with easing

イージング付きのモーフィング



17-silhouette_morphing-B.hiplc

The previous linear morphing example lacks the controllability of point sorting as well as the smoothness of the morphing. If linear morphing is not good enough for you for visual needs then why not use VEX for more controllability.

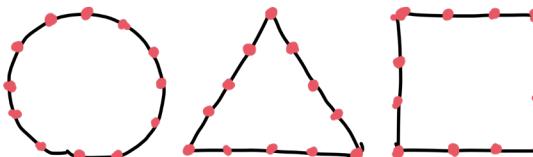
In this example in order to get more smooth morphing the order of points for each silhouette is modified by smoothing out the original silhouette until it becomes convex shape and use points' angle measured from silhouette's center to sort them out. It also has an easing function when it morphs from one shape to another by using VEX.



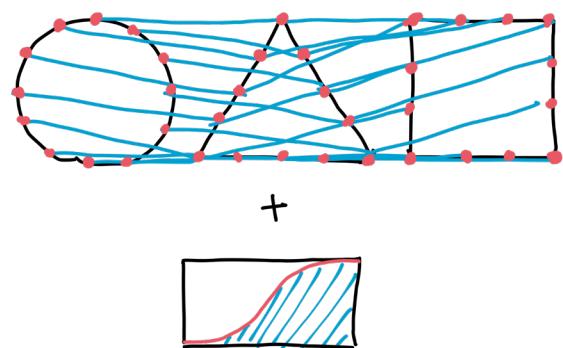
前の線形モーフィングの例はポイントの並び替えやモーフィングの滑らかさのコントロールができない形でした。もし線形のモーフィングだけでは役不足と感じた場合、VEXを使ってみるのはいかがでしょう。

この例では、より滑らかなモーフィングを得るために各シルエットのポイントの順番を、シルエットが凸形状になるまでスムージングしてから、シルエットの中心から見た各ポイントの角度に応じてポイントを並び替えていきます。また、モーフィングをVEXで行い、その際ひとつの形状から別の形状へモーフィングする際イージングの効果を加えています。

①

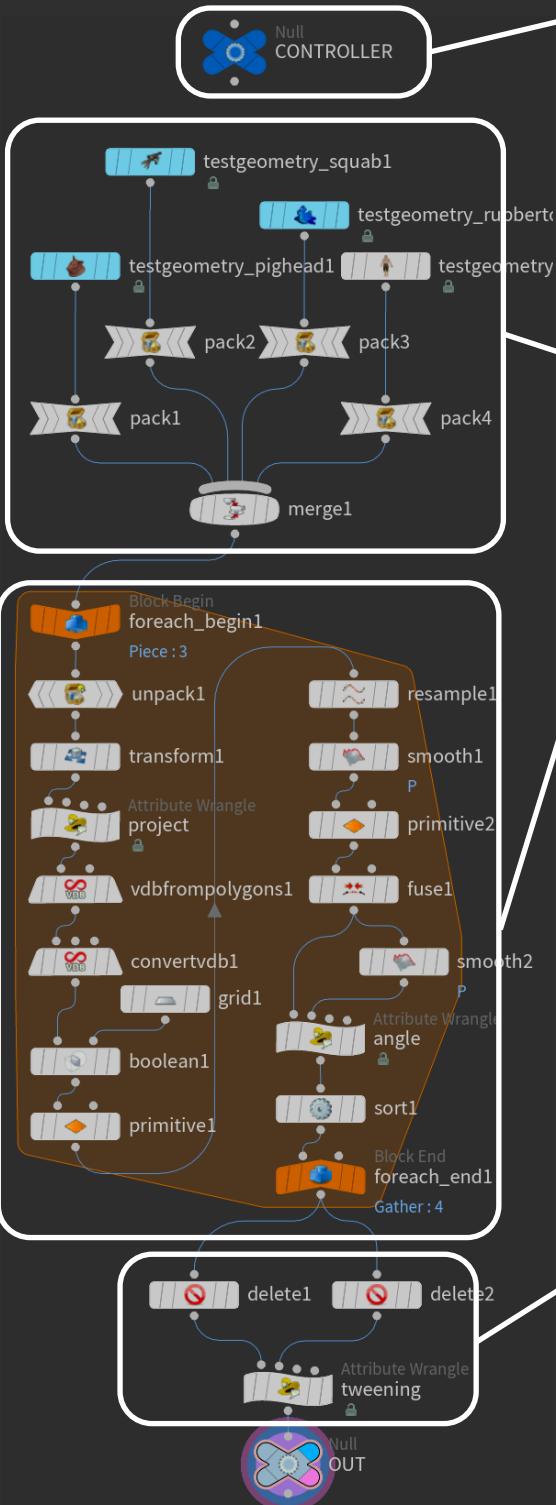


②



Prepare three silhouette outlines with same number of points.
同じ数だけのポイントをもった三つのシルエットのアウトラインを作る。

Create a morphing with easing between three silhouette outlines using VEX with ramp parameter.
VEXとrampパラメータを利用して三つのシルエットのアウトライン間でイージング付きsのモーフィングを作る。



Setup parameters

- **tween:** morphing time
- **tween ramp:** ramp curve for easing

パラメータの設定

- **tween:** モーフィングの時間
- **tween ramp:** イージングのためのランプ曲線

Prepare multiple shapes

Prepare multiple shapes in order to create silhouette.

複数の形状を用意する

シルエット作成のために複数の形状を用意します。

Create silhouette and sort points

Create silhouette from each shape using VDB together with Boolean SOP and sort the order of points by making silhouette round convex shape to get the circular angle.

シルエットを作りポイントを並び替える

個々の形状からシルエットをVDBとBoolean SOPを利用して作成し、ポイントの順番をシルエットを丸い凸形状にしてポイントの円形角度を得ることで並び替えています。

Create morphing with easing

Create morphing with easing function using VEX from source and target shape.

イージング付きのモーフィングを作る

VEXを使ってソースからターゲットの形状へモーフィングをイージング付きで作ります。

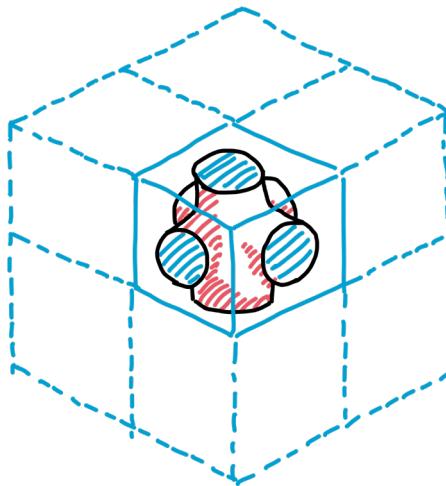
18. Isosurface lattice アイソサーフェスのラティスを作る

Create a lattice structure using isosurface as an element like an image next page. If possible try to create an unique lattice structure using your choice of mathematical equation.

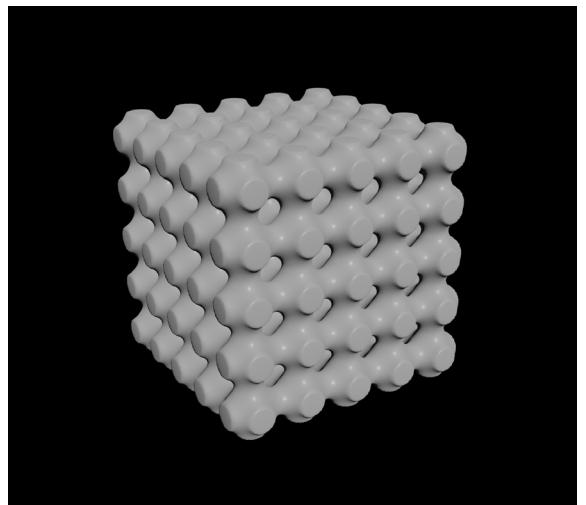
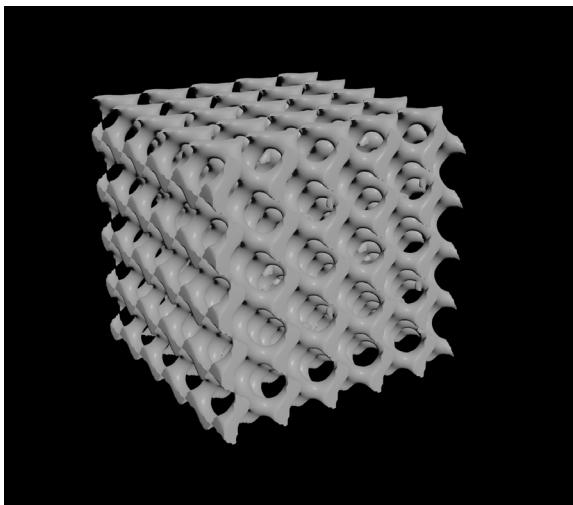
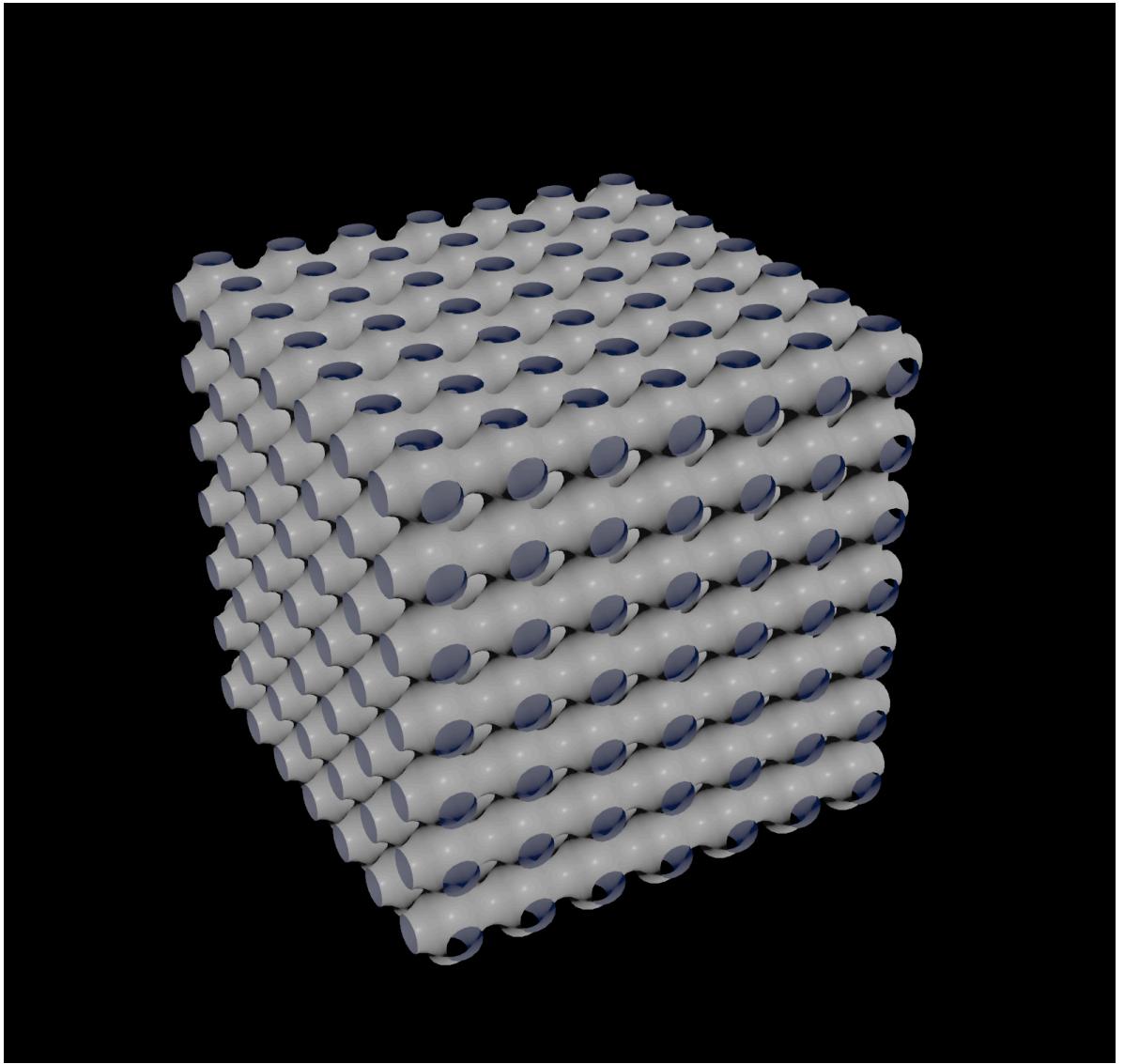
次のページの図のようなアイソサーフェスを要素として使ったラティス構造を作ってください。可能であれば自分の好きな数式を探し出し、それを使ってラティス構造を作ってみてください。

Preferred parameters / 推奨パラメータ:

- 3d grid resolution / 3次元グリッドの解像度
- Shape transformation threshold / 形状変形閾値



Create lattice structure using isosurface.
アイソサーフェスを使ってラティス構造を作る。



18-A. Lattice using metaball

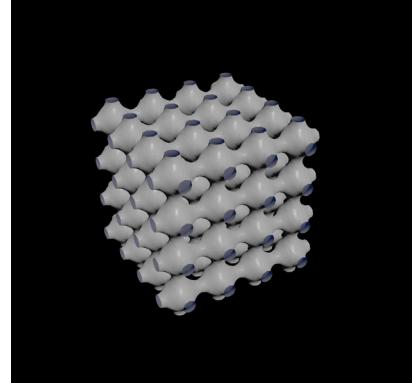
メタボールを使ったラティス



18-lattice-A.hiplc

The easiest way to create isosurface is to use metaball which is one type of isosurface. Placing metaballs on each vertices of a box then trimming this shape with boolean with the same size box, you can have a basic shape which can be placed on a 3d grid seamlessly. When you do that you could construct a lattice structure.

アイソサーフェスを作る最も簡単はアイソサーフェスの種類の一つであるメタボールを利用することです。メタボールをボックスの各頂点に配置し、同じサイズのボックスでブーリアンを使って切り取ることで、それをベース形状として三次元のグリッドにシームレスに配置することができるようになります。結果アイソサーフェスのラティス構造を作ることができます。

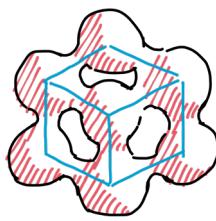


①



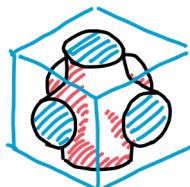
Create a box as a boundary for lattice element.
ラティス構造の要素のためのバウンダリをボックスとして作る。

②



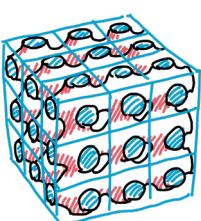
Place metaball on every vertices of the box.
ボックスの頂点にメタボールを配置する。

③



Trim a metaball shape with a box to create a lattice structure element.
メタボールの形状をボックスでトリムしてラティス構造の要素を作る。

④



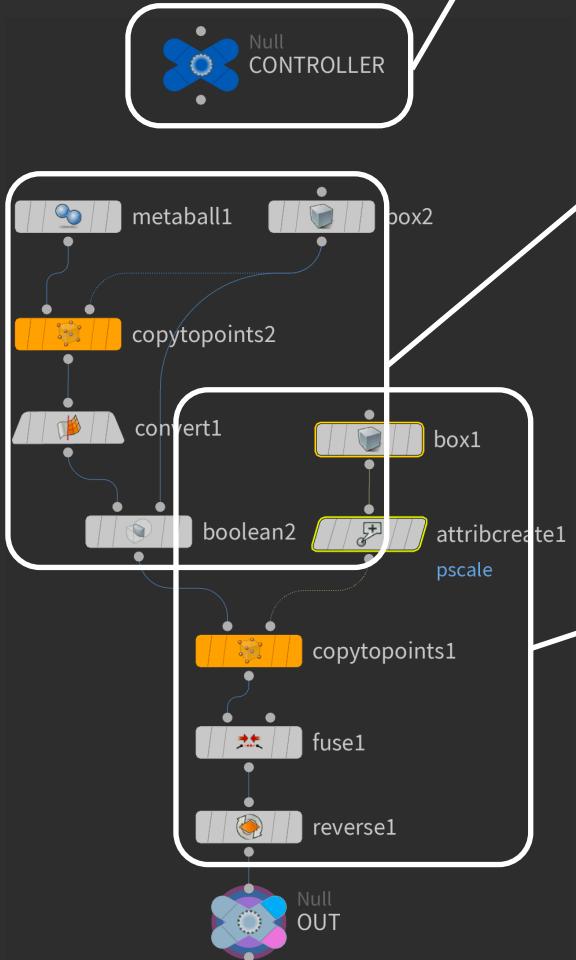
Copy lattice element on a 3d grid to create lattice structure.
ラティス構造の要素を三次元グリッド上に配置してラティス構造を作る。

Setup parameters

- **grid_num:** number of grid cells
- **cell_size:** grid cell size
- **metaball_size:** size of metaball

パラメータの設定

- **grid_num:** グリッドセルの数
- **cell_size:** グリッドセルのサイズ
- **metaball_size:** メタボールのサイズ



Create lattice element

Create seamless lattice element using metaball.

ラティス構造の要素を作る

シームレスにつながるラティス構造の要素をメタボールを使って作ります。

Complete lattice structure

Create lattice structure by copying lattice element you made with metaball on a 3d grid points seamlessly.

ラティス構造を完成させる

ラティス構造の要素を三次元グリッドのポイントの位置にシームレスに配置することでラティス構造を完成させます。

18-B. Lattice using volume

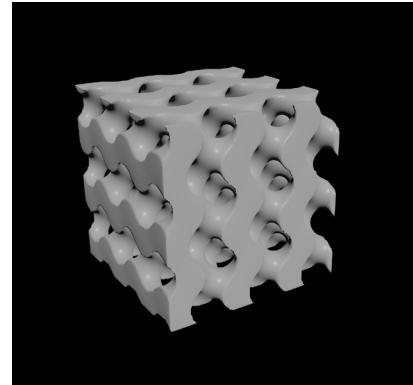


ボリュームを使ったラティス

18-lattice-B.hiplc

Using just metaball for isosurface might not be enough when you want to have more control over the shape since there are so many types of shapes exist when it comes to isosurface.

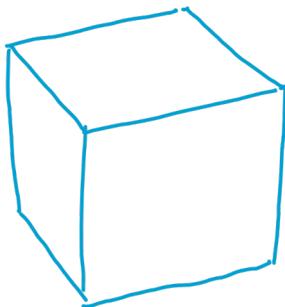
When you want to explore various interesting iso-surface shape it's a good idea to deal with volume using some mathematical equations. In this example Volume Wrangle is used to create two types of iso-surface by using simple trigonometry equations.



もっとアイソサーフェス形状のコントロールを行ないたい場合、メタボールだけでは役不足かと思います。アイソサーフェスと一言で言っても、無数のタイプの形状が存在しているからです。

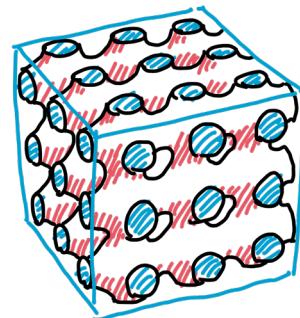
この無数の形状の可能性を探索したい場合、いくつかの式とボリュームを利用することが得策です。この例では、Volume Wrangleと簡単な三角関数を利用して二種類のアイソサーフェスを作っています。

①

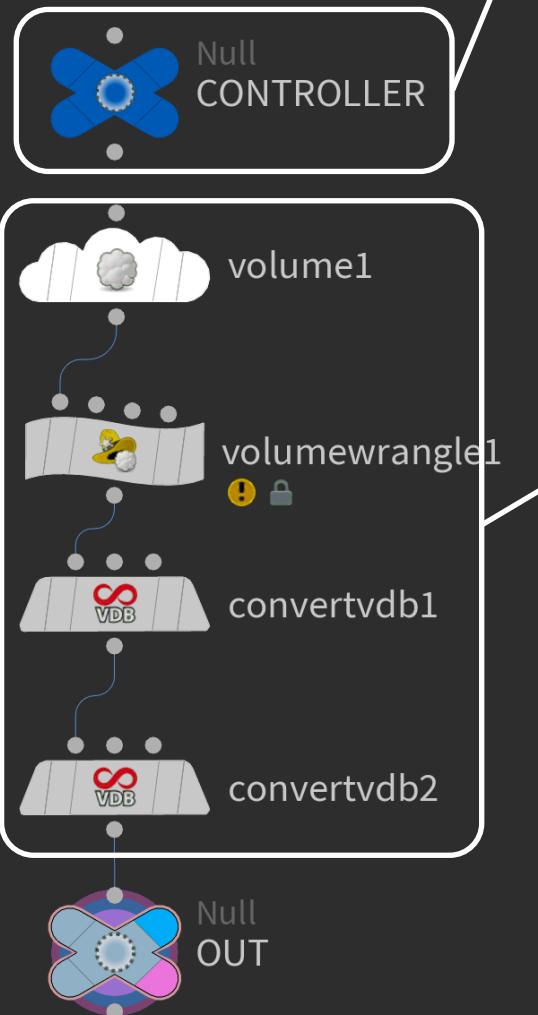


Create volume as a boundary for lattice structure.
ラティス構造の領域としてボリュームを作る。

②



Use Volume Wrangle with mathematical equation to create lattice structure from the volume.
式をVolume Wrangleと利用して、ボリュームからラティス構造を作る。



Setup parameters

- **grid_num:** number of grid cells
- **switch_shape:** shape type
- **shape:** isosurface threshold

パラメータの設定

- **grid_num:** グリッドセルの数
- **switch_shape:** 形状のタイプ
- **shape:** アイソサーフェスの閾値

Create isosurface with volume

Use Volume Wrangle SOP to create seamless lattice structure using isosurface. Specifically calculate density value for each voxel using following mathematical equations.

ターゲットの配置

Volume Wrangle SOPを利用してシームレスなラティス構造をアイソサーフェスで作ります。具体的には、ボリュームの各ボクセルの密度を次のような計算式で計算することで作ります。

[type 1 / タイプ 1]

$$\cos(x) + \cos(y) + \cos(z)$$

[type 2 / タイプ 2]

$$\cos(x)*\sin(y) + \cos(y)*\sin(z) + \cos(z)*\sin(x)$$

19. **Braid rope** 三編みロープ

Create a braid rope with length and rope thickness parameter. If possible try to create a braid rope along a free 3d curve.

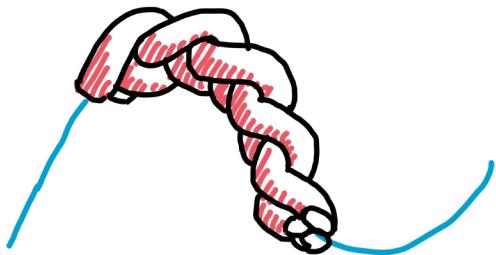
長さと太さのパラメータをもった三編みロープを作ってください。可能であれば、自由な曲線に沿った三編みロープを作ってください。

Preferred parameters / 推奨パラメータ:

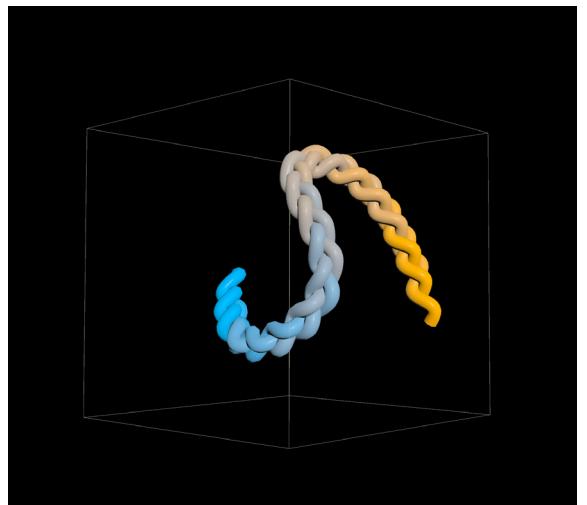
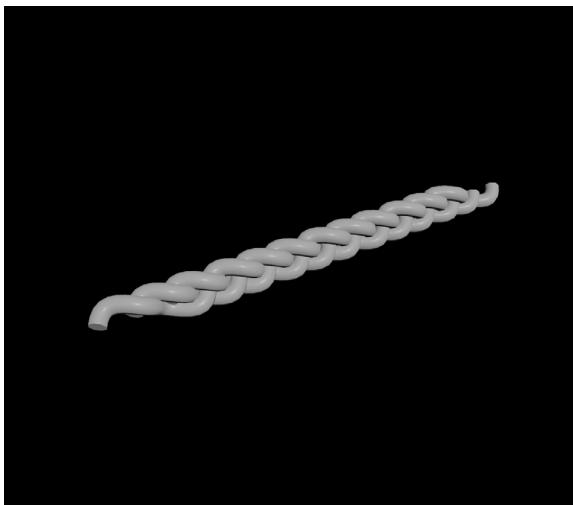
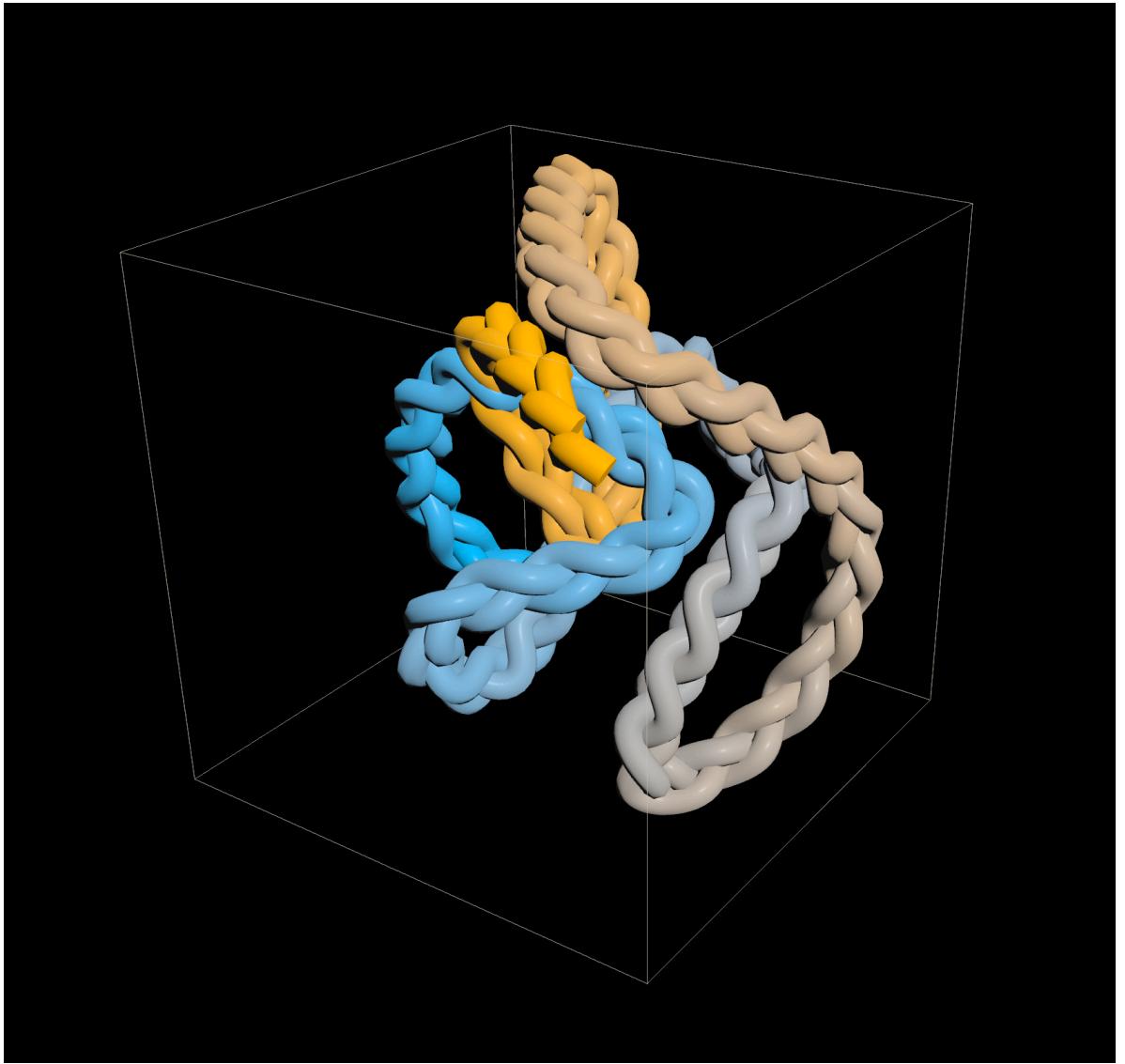
- Length of a braid rope / 三編みロープの長さ
- Thickness of a braid rope / 三編みロープの太さ



Create straight braid rope.
直線的な三編みロープを作る。



Create a braid rope along a curve.
曲線に沿った三編みロープを作る。



19-A. Straight braid rope

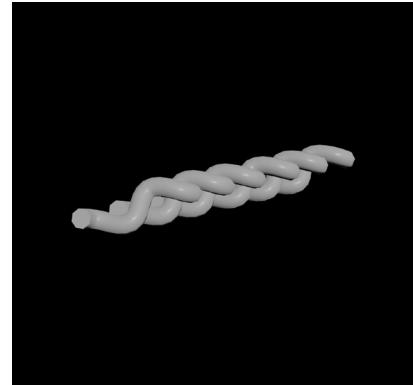
直線的な三編みロープ



19-braid-A.hiplc

This example creates a straight braid rope by preparing three identical wavy curve made with trigonometry and repeatedly copied horizontally to create long continuous braid rope.

この例では、三角関数を利用して作られた三つの同じ形の波打つ曲線をまず用意し、それを水平にシームレスに繋いでいくことで直線的な三編みロープを作ります。



①



②



③



Create a base three curves for braid rope.

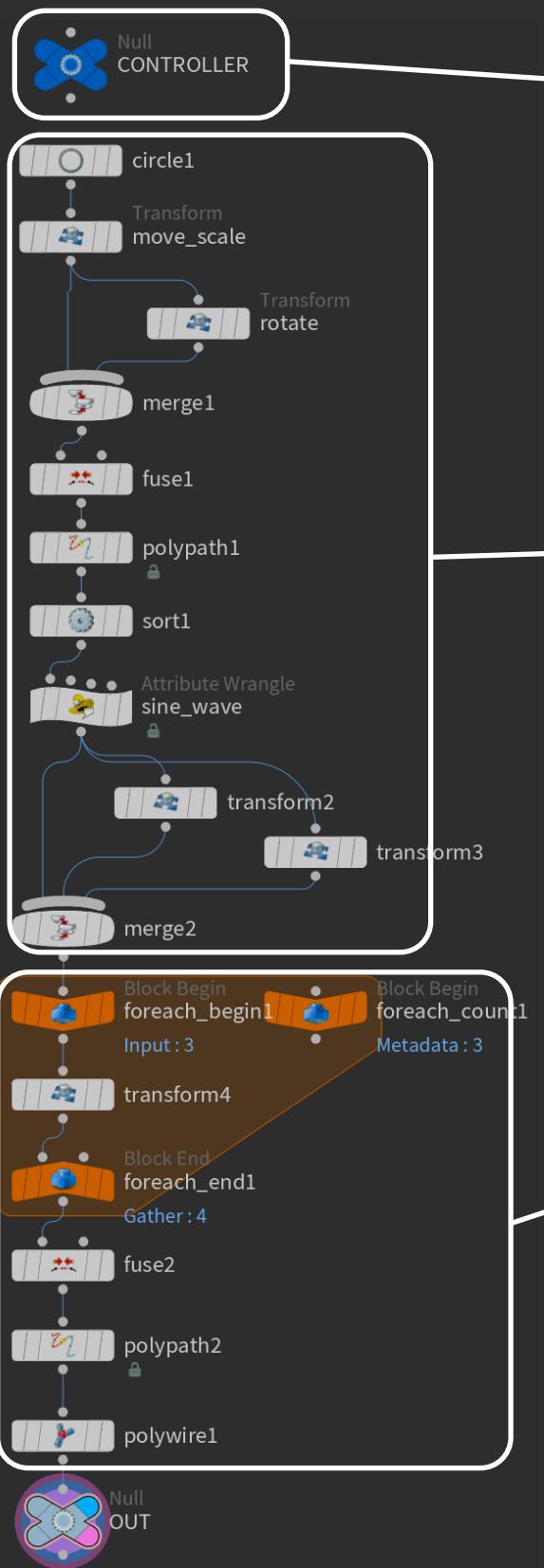
三編みカーブのベースとなる三つのカーブを作る。

Connect base curves horizontally.

三編みロープのベースの曲線を水平につなげる。

Thicken curves to create braid rope using PolyWire SOP.

PolyWire SOPを利用して曲線に厚みを与えて三編みロープを作る。



Setup parameters

- **length:** length of braid rope
- **thickness:** thickness of rope

パラメータの設定

- **length:** 三編みロープの長さ
- **thickness:** ロープの厚さ

Create three wavy curves

Create three identical wavy curves offsetted horizontally using sine function with VEX.

三つの波打つ曲線を作る

水平にオフセットした三つの同じ形の波打つ曲線をVEXのsine関数を利用して作ります。

Create continuous braid rope

Create straight continuous braid rope by copying base three wavy curves horizontally. Then you can use PolyWire SOP to thicken the rope.

つながった三編みロープを作る

先に作った三つのベースの曲線を水平につなげることで、直線的につながった三編みロープを作ります。そしてPolyWire SOPを使ってロープに厚みをつけます。

19-B. Braid rope along curve

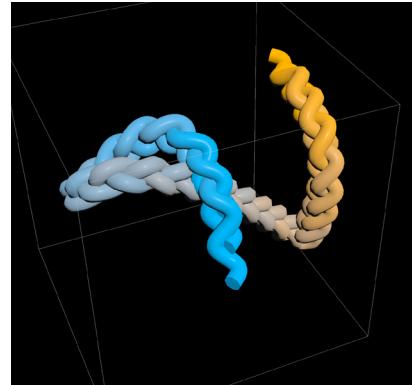
曲線に沿った三編みロープ



19-braid-B.hiplc

It is not too flexible if the rope can only be made straight but you might want to create a braid rope along any free curve.

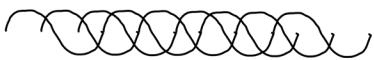
This example creates a braid rope along a curve by using a fair amount of VEX code to remap the straight braid rope onto a free drawn curve instead of using Wire Deform SOP. The reason why this example is not using Wire Deform SOP is that this node cannot avoid the twisting of a free drawn curve but using VEX can avoid it by referring the clean up vector from a geometry created by Poly-Wire SOP.



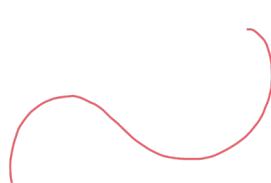
直線的な三編みロープでは自由度が低いのは明白で、自由な曲線上に三編みロープを作りたい場合もあると思います。

この例では、そこそこの量のVEXコードを書いて直線的な三編みロープを自由な曲線上にマッピングしています。その際、同じようなことを比較的簡単にできるWire Deform SOPは、自由曲線のねじりの影響を受けきれいにマッピングすることができないため使っていません。その代わり、曲線からパイプ形状を作る際にはねじりなしで綺麗に作ってくれるPolyWireのジオメトリをVEXで参照し、upベクトルを作ることでねじりのない三編みロープのマッピングを実現しています。

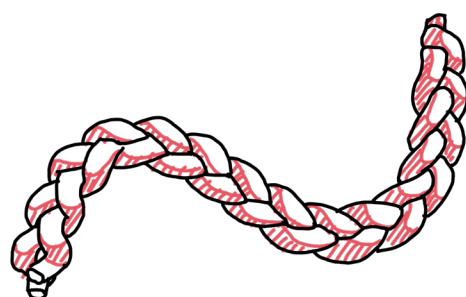
①



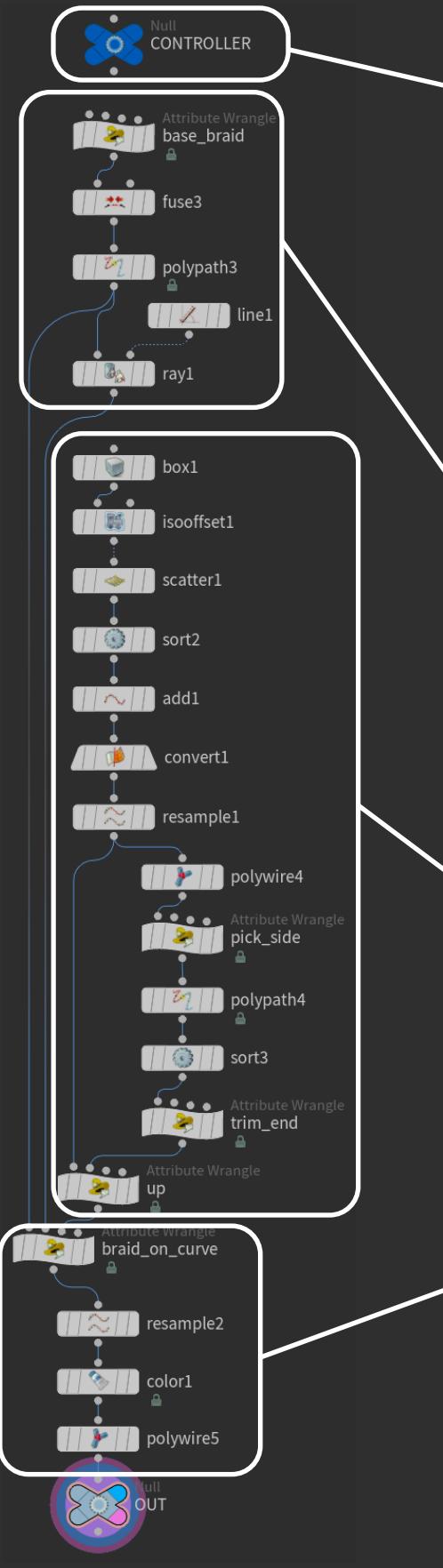
②



Create straight braid rope curves and a free curve you want to map braid rope to.
直線的な三編みロープの曲線と、三編みロープをマップするための自由曲線を作る。



Map straight braid rope curves on a free curve to create braid rope using VEX.
VEXを使って直線的な三編みロープを曲線上に沿わせる。



Setup parameters

- **pt_num:** number of points for free curve
- **seed:** random seed for curve shape
- **length:** length of a free curve
- **thickness:** thickness of rope

パラメータの設定

- **pt_num:** 自由曲線を構成するポイントの数
- **seed:** 自由曲線の形状のランダムシード
- **length:** 自由曲線の長さ
- **thickness:** ロープの厚さ

Create straight braid rope curve

First create a straight braid rope curve mostly using VEX with sine wave.

直線的な三編みロープの曲線を作る

まず直線的な三編みロープの曲線をVEXと
サイン関数を使って作ります。

Get untwisted up vector

Get untwisted up vector by referring to geometry created by PolyWire SOP using VEX.

ねじっていないupベクトルを取得する

VEXを使ってPolyWireによって作られたジオメトリを参照することで、ねじりのないupベクトルを取得します。

Map straight braid to free curve

Use VEX to map straight braid rope curve to free curve without twist by using previously retrieved up vector.

直線的な三編みを自由曲線にマップする

VEXを使って、事前に取得したupベクトルを利用することでねじりのない形で直線的な三編みロープの曲線を自由曲線にマップします。

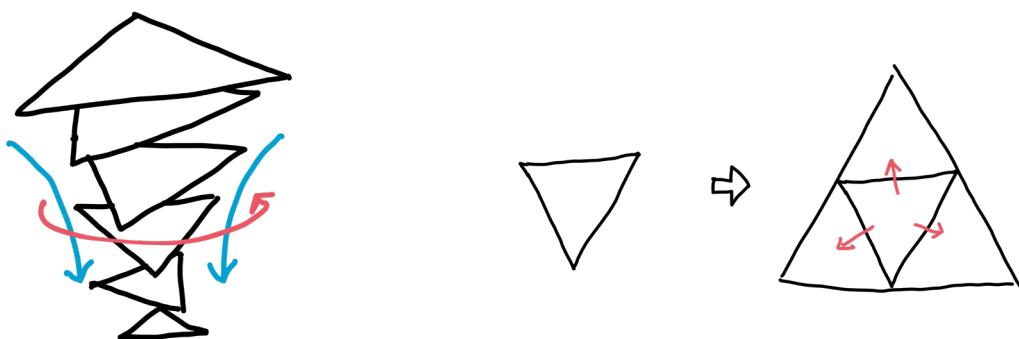
20. Fractal column フラクタルな柱を作る

Create a fractal column like an image shown next page. The rule is to first prepare a polygon with specified side numbers. Then copy it to apply a transformation like rotation, scaling, and translation to create a next column layer. Repeat this procedure using the last created column layer to create a gradually transforming column. If possible create a column branch based on some iteration timing.

次のページの図にあるようなフラクタルな柱を作ってください。ここでのルールは、まず最初に辺の数を決めたポリゴンを用意します。その上で、そのポリゴンをコピーして回転や拡大縮小、移動といった変形をして、それを柱の次のレイヤーとします。このプロセスを最後に作られたレイヤーに対して繰り替えしを行い、グラデーションアルに変形するフラクタルな柱を作ってください。可能であれば、任意の繰り返しステップのタイミングに応じて柱が枝分かれするようにしてください。

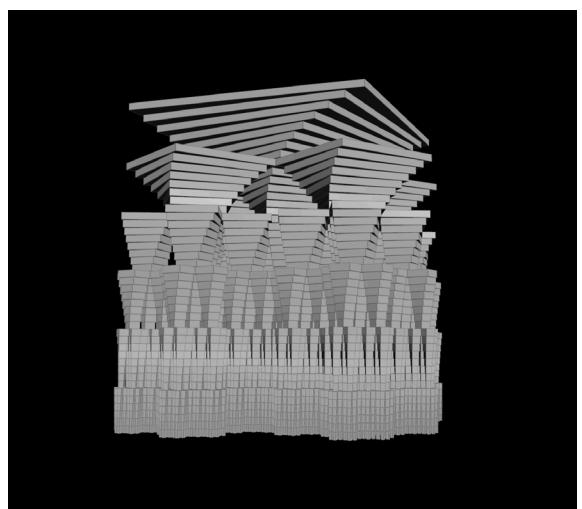
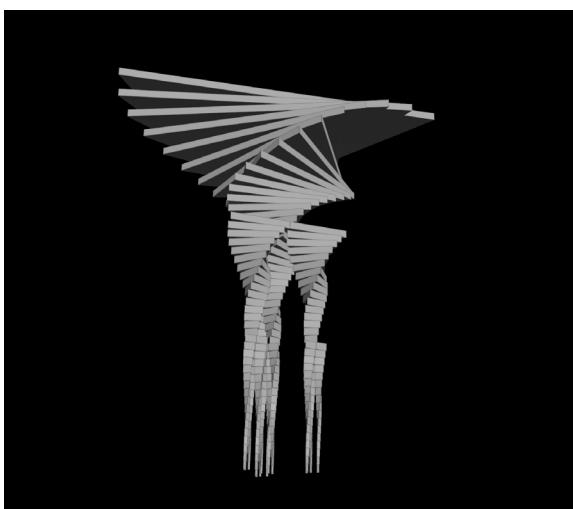
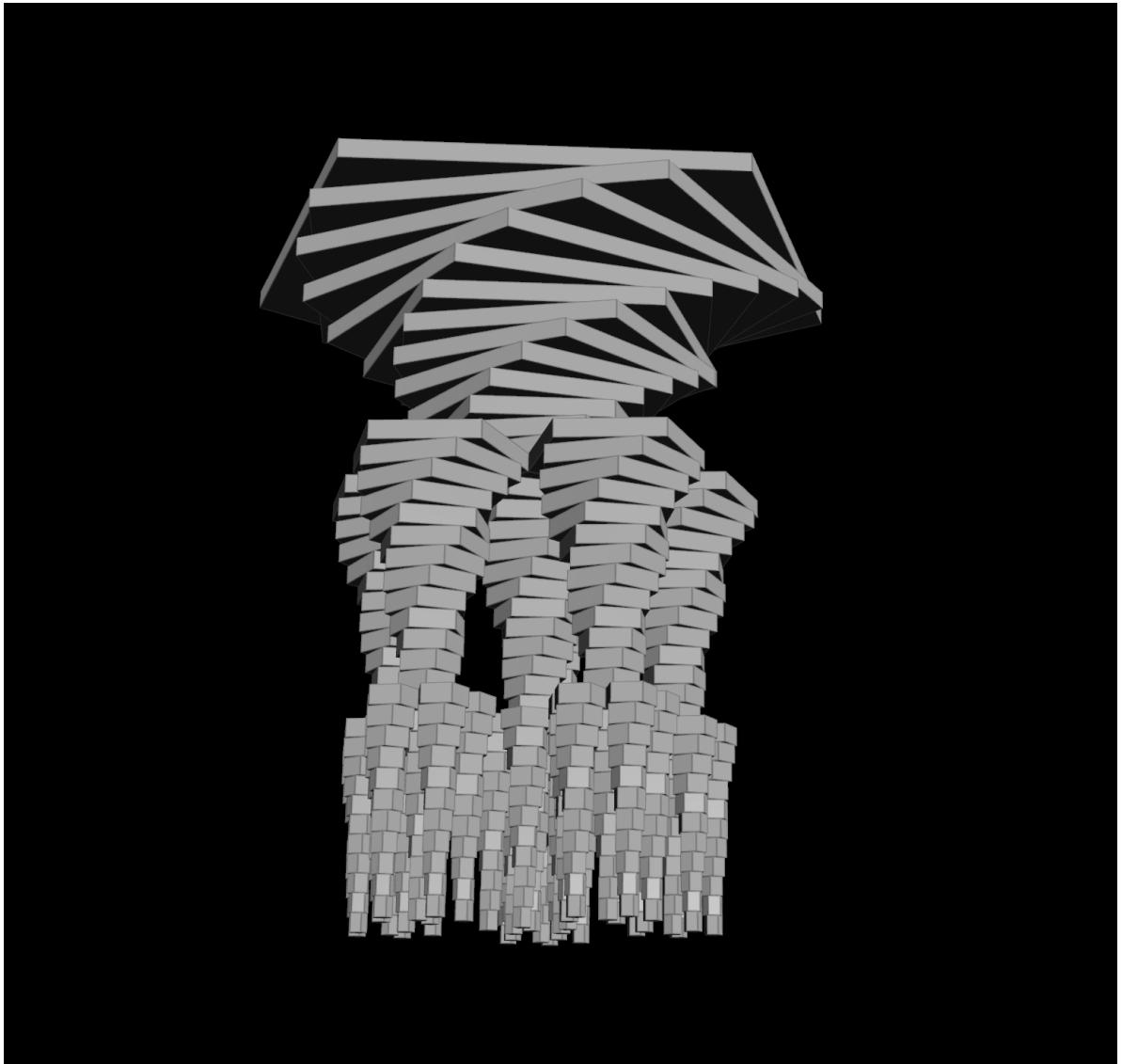
Preferred parameters / 推奨パラメータ:

- Number of sides for base polygon / ベースのポリゴンの辺の数
- Number of layers for column / 柱のレイヤーの数
- Transformation value for each iteration / ステップごとの変形値
- Branching timing / 柱の枝分かれのタイミング



Create fractal column move / scale / rotate transformation.
移動・拡大縮小・回転の変形を使ってフラクタルの柱を作る。

Split surface at every some recursive iteration.
任意の数の再帰処理毎にサーフェスを分割する。



20-A. Fractal using feedback loop

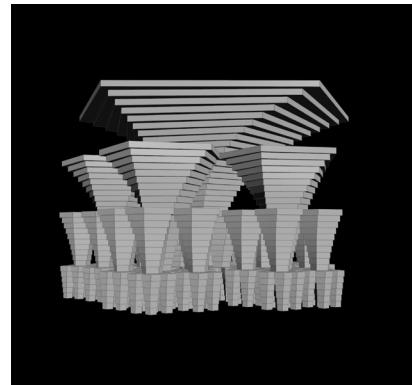
フィードバックループを利用したフラクタル



20-fractal_column-A.hiplc

In order to create a fractal geometry there are several ways to do it in Houdini, and one of the way to achieve this is to use For Each loop with Feedback option. One of situation you want to use feedback type loop for fractal geometry is when you want a still geometry when you don't need animation since all the calculation is done in one frame.

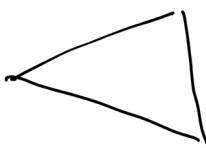
This example is using polygon with specified number of sides as a base shape and using For Each Number loop with feedback option to have a fractal column. It also uses nested loop together with Switch SOP so that the column plan will split every some iterations.



フラクタルなジオメトリをHoudiniで作るにはいくつかの方法がありますが、ひとつ的方法として考えられるのはFor Eachループをフィードバックのオプション付きで利用することです。この手法を使いたい場面としては、アニメーションの必要のない一フレームで完成した形状を作りたい時などが考えられます。

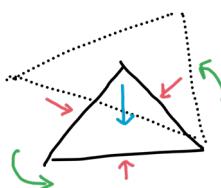
このサンプルでは辺の数を指定可能なポリゴンをベースの形状として使い、フィードバックのオプション付きのFor Each Numberループを利用してフラクタルな柱を作っています。ループの中に別のループやSwitch SOPを利用することで、複数のループ処理毎に柱が分割されるようにしています。

①



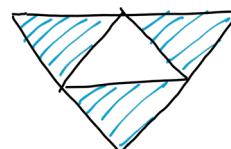
Prepare base polygon shape.
ベースのポリゴン形状を用意する。

②



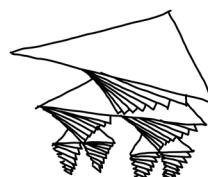
Transform base polygon inside For Each loop with feedback option.
フィードバックのオプション付きのループの中でベースのポリゴンを変形する。

③

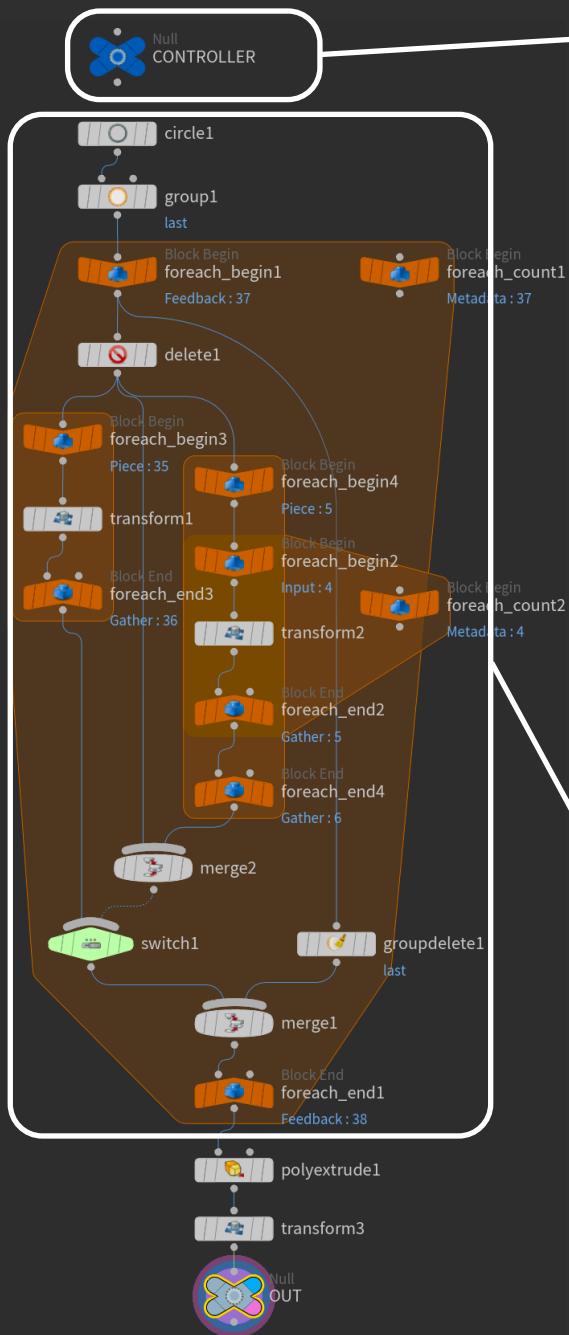


Split surface at every some loop iteration.
任意の数のループ処理毎にサーフェスを分割する。

④



Continue Step 2~3 to create fractal column.
ステップ2~3を繰り返してフラクタルの柱を作る。



Setup parameters

- **base_size**: size of base polygon
- **step_angle**: step angle value for rotation
- **step_scale**: step scale value for scaling
- **branch_timing**: column branching timing
- **iteration**: number of loop iteration
- **shape**: number of polygon sides

パラメータの設定

- **base_size**: ベースのポリゴンのサイズ
- **step_angle**: ステップ毎の回転角度
- **step_scale**: ステップ毎の縮尺値
- **branch_timing**: 分割のタイミング
- **iteration**: ループの回数
- **shape**: ポリゴンの辺の数

Create fractal column with loop

Use For Each Number loop with feedback option to create a fractal column starting from a simple polygon. At each loop iteration the last created polygon is copied and rotated / scaled / moved to be stacked as a column and at every specified iterations the column will split.

ループを使ってフラクタルの柱を作る

フィードバックのオプション付きのFor Each Numberループを使い、ベースとなるシンプルなポリゴン形状から始めてフラクタルの柱を作ります。毎ループ処理において、最後に作られたポリゴンに対して回転・拡大・縮小・移動を行ない柱を積み上げて行きます。そして任意のループ回数毎に柱を分割します。

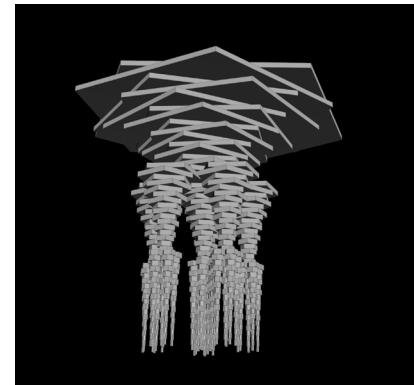
20-B. Fractal using solver

ソルバーを使ったフラクタル



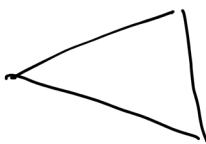
20-fractal_column-B.hiplc

Another popular way to create fractal geometry in Houdini is to use Solver which is pretty similar to feedback loop, only difference is that the calculation done in the loop is done per frame when you use Solver which makes it possible to easily create a generative animation. In this example Solver is used to create fractal column instead of For Each loop but rest of the process is pretty much the same with previous example.

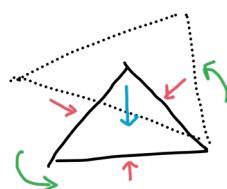


フラクタルなジオメトリを作る上でもう一つのよく使われる手法がフィードバックタイプのループと良く似ているSolver(ソルバー)を利用する手法です。ソルバーを使う手法がループを使う手法と異なるのは、ループの中で行っていた計算をフレーム単位で行うという点です。これにより、For Eachループと比べて容易にフラクタルの生成アニメーションを作ることができます。この例では、For Eachループの代わりにソルバーを使っていますが、それ以外は前の例とほぼ同じ内容となっています。

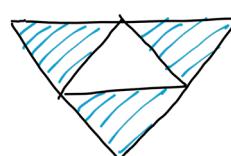
①



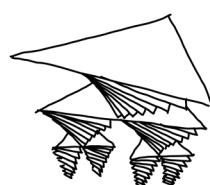
②



③



④

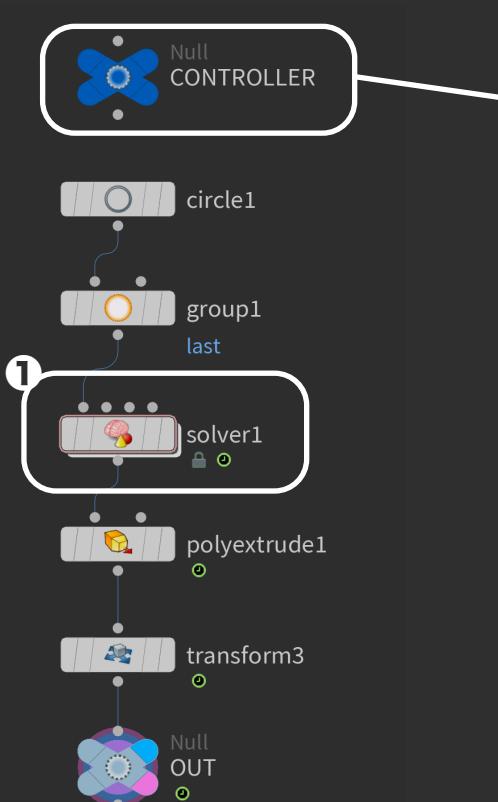


Prepare base polygon shape.
ベースのポリゴン形状を用意する。

Transform base polygon inside Solver network.
ソルバーの中でベースのポリゴンを変形する。

Split surface at every some frames.
任意のフレーム数毎にサーフェスを分割する。

Continue Step 2~3 to create fractal column.
ステップ2~3を繰り返してフラクタルの柱を作成する。

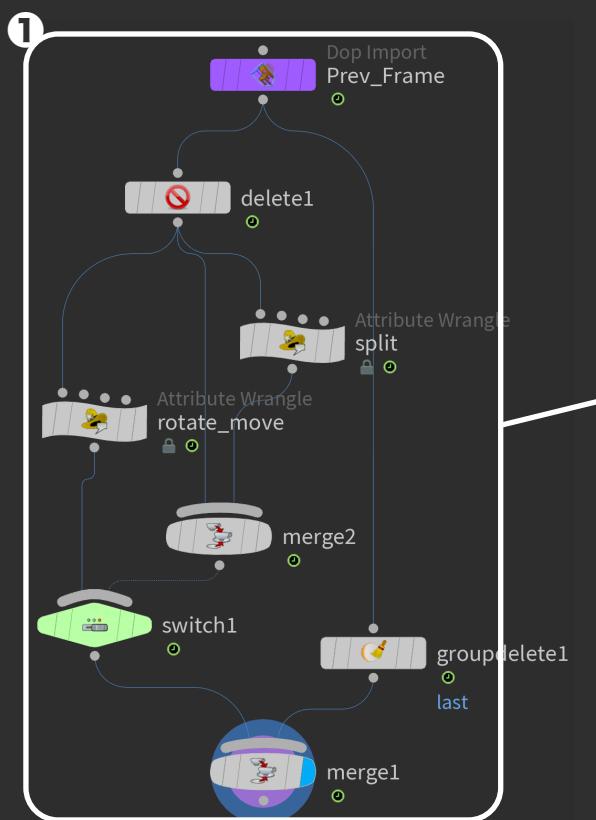


Setup parameters

- **base_size**: size of base polygon
- **step_height**: step height value
- **step_angle**: step angle value for rotation
- **step_scale**: step scale value for scaling
- **branch_timing**: column branching timing
- **shape**: number of polygon sides

パラメータの設定

- **base_size**: ベースのポリゴンのサイズ
- **step_height**: ステップ毎の高さ
- **step_angle**: ステップ毎の回転角度
- **step_scale**: ステップ毎の縮尺値
- **branch_timing**: 分割のタイミング
- **shape**: ポリゴンの辺の数



Create fractal column with solver

Use Solver SOP to create a fractal column starting from a simple polygon. At each frame the last created polygon is copied and rotated / scaled / moved to be stacked as a column and at every specified frames the column will split.

ソルバーを使ってフラクタルの柱を作る

Solver SOPを使い、ベースとなるシンプルなポリゴン形状から始めてフラクタルの柱を作ります。毎フレームにおいて、最後に作られたポリゴンに対して回転・拡大縮小・移動を行ない柱を積み上げて行きます。そして任意のフレーム数毎に柱を分割します。

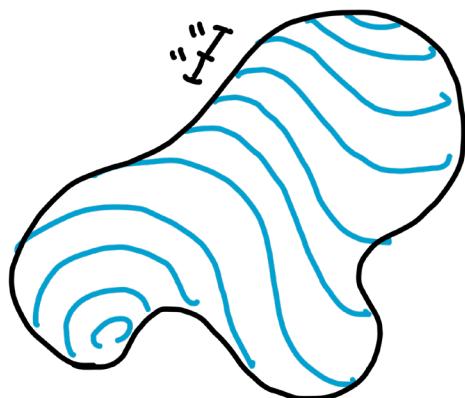
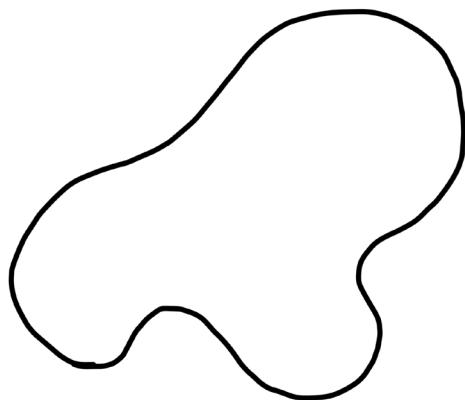
21. Surface marbling サーフェスにマーブル模様を作る

Create a marbling pattern on any input geometry. The marble pattern here should be created as a collection of curves drawn on a surface and a distance between neighbor curves should be always the same and be modifiable as a parameter.

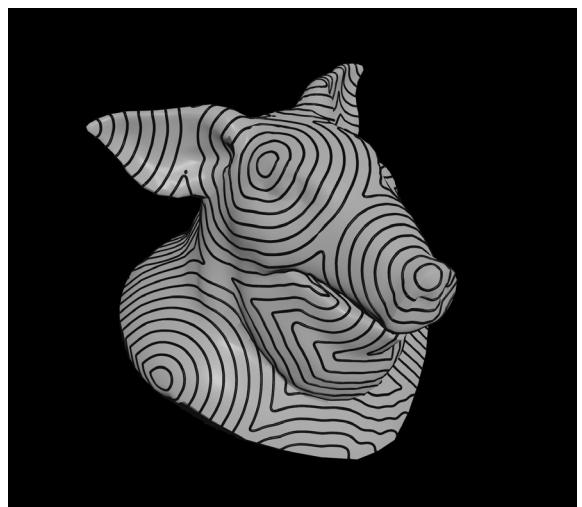
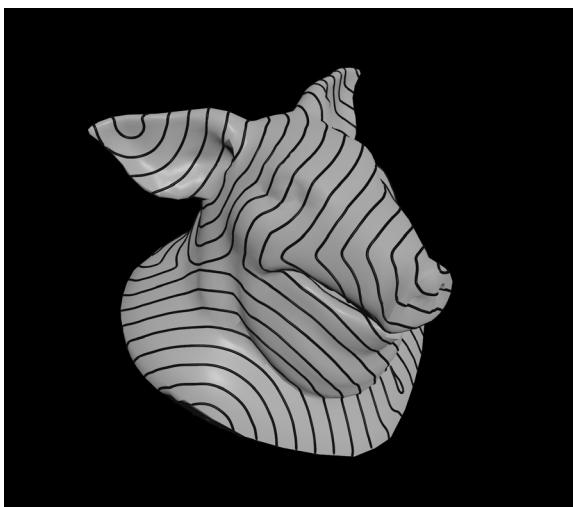
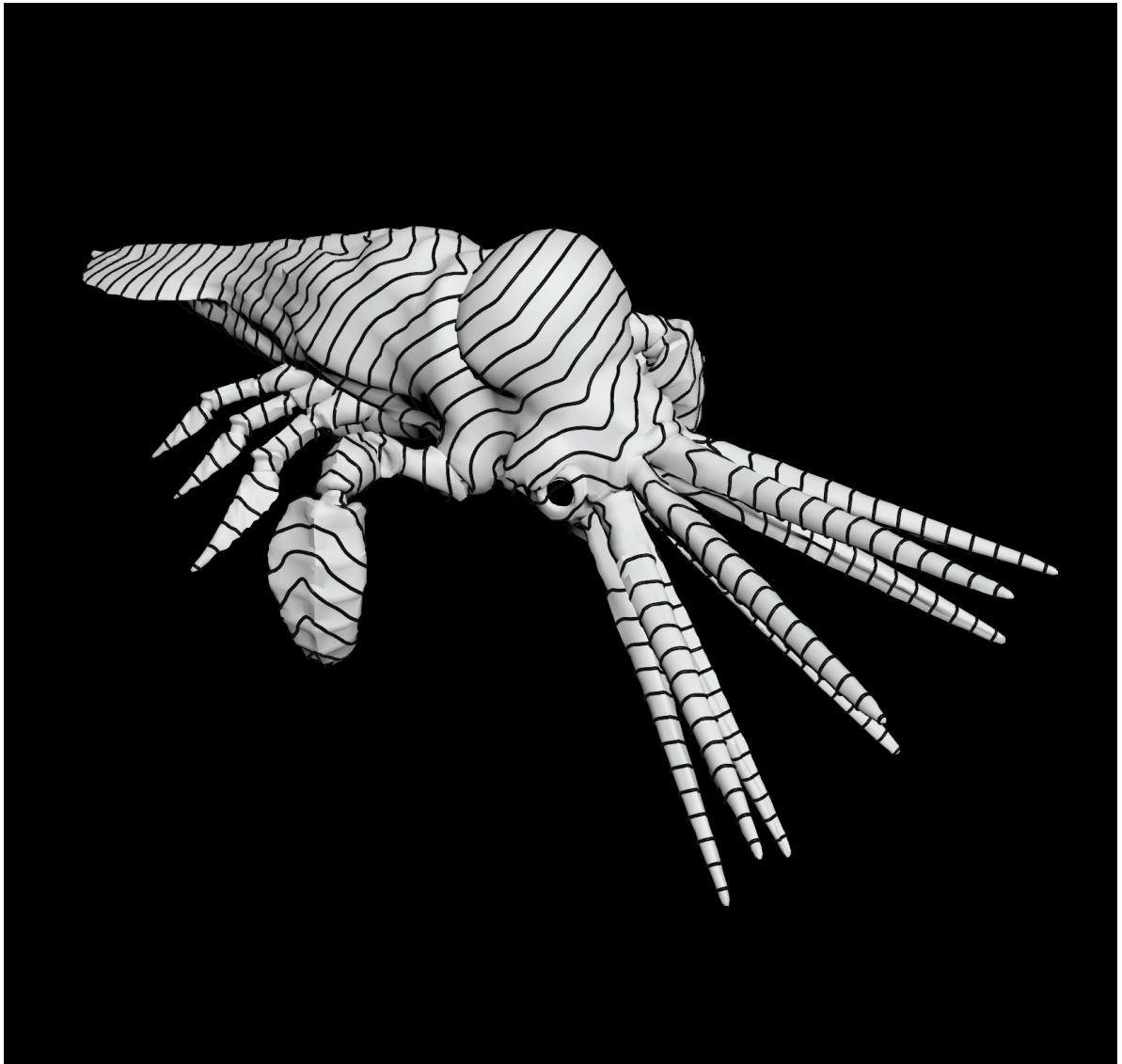
任意のジオメトリの上にマーブル模様を作ってください。ここで言うマーブル模様とは、サーフェスの上に乗った複数の曲線によって作られた模様のことを指します。その時、隣り合う曲線の間の距離は常に同じであることが条件となります。この時この距離はパラメータとして変更できるようにしてください。

Preferred parameters / 推奨パラメータ:

- Distance between neighbor curves / 隣り合う曲線の距離
- Number of starting point for pattern / 模様が始まる点の数



Create marbling pattern on any geometry.
任意のジオメトリにマーブル模様を作る。



21-A. Marbling with loop

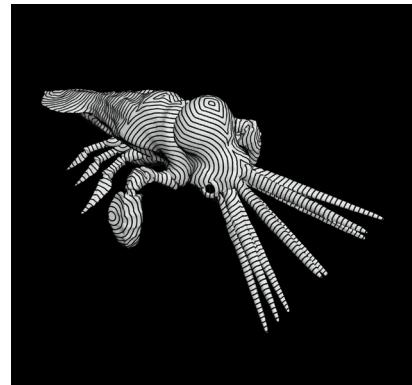
ループを使ったマーブル模様



21-surface_marbling-A.hiplc

Similar to fractal geometry the method used for this example is a feedback loop. Feed back loop is useful for recursive calculation which includes fractal and the marbling pattern can also be made using this recursive calculation.

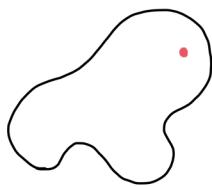
This example starts by retrieving naked edges from an input geometry by cutting it using spheres. Then by using For Each feedback type loop you recursively create a pipe geometry from the naked edge which is again used to cut the geometry to create another naked edge. In the end you will cover the base input geometry with marbling like contour curves.



フラクタル形状と同じように、この例ではフィードバックループを使っています。フィードバックループはフラクタルを含めた再帰計算が必要な形状に適しています。今回作るマーブル模様にも再帰計算を利用するのが最適です。

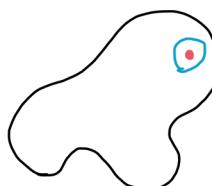
この例ではまずベースのジオメトリを複数の球体で切り抜き、それによって生まれたサーフェスのオープンエッジを取得するところから始めます。その上で、フィードバックタイプのFor Eachループを使い、ループの中でそのオープンエッジからパイプを作り、ベースのジオメトリをそのパイプで切り取ります。そうすることで新たなオープンエッジを得られ、それを再帰的に繰り返すことで最終的にはジオメトリ全体をサーフェス上等距離のセンター曲線で埋めることができます。

①



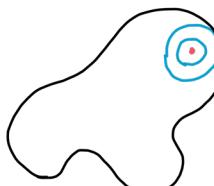
Create an initial point on a geometry.
ジオメトリの上に初期のポイントを作る。

②



Place sphere on a point and trim geometry with it to get naked edge.
ポイントに球体を配置し、それを使ってジオメトリを切り取り、オープンエッジを取得する。

③

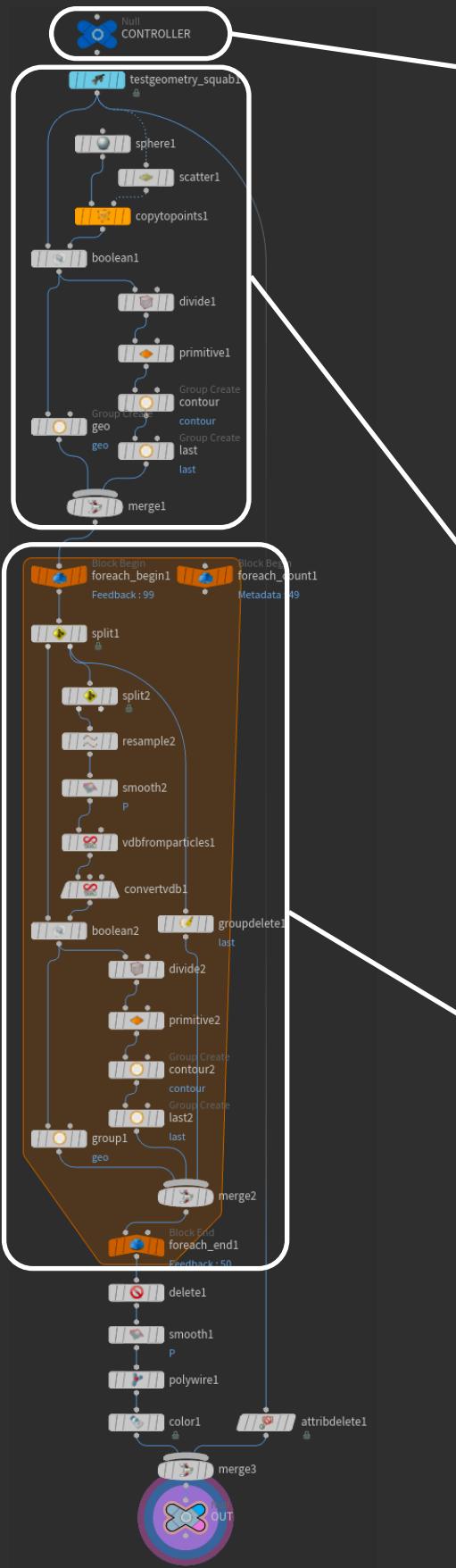


Create a pipe shape from a naked edge and trim geometry with it to get new naked edge.
オープンエッジからパイプを作り、それを使ってジオメトリを切り取り、新たなオープンエッジを得する。

④



Continue Step 3 to create marble pattern on the geometry by using For Each loop with feedback option.
フィードバック付きのループを使ってステップ3を繰り返し行ない、ジオメトリの上にマーブル模様を作る。



Setup parameters

- thickness:** distance between contours
- max_contour:** max number of contours
- pt_num:** number of initial spheres
- seed:** random seed for sphere position

パラメータの設定

- thickness:** コンター間の距離
- max_contour:** 最大コンター数
- pt_num:** 初期球体の数
- seed:** 初期球体の位置のランダムシード

Get initial naked edges

Get initial naked edges by cutting geometry using number of spheres.

初期オーブンエッジを作る

初期のオーブンエッジを、ジオメトリを複数の球体で切り取ることで取り出します。

Recursively create contour

Use feedback loop to recursively create contour curve on geometry by cutting geometry using pipe generated from last created naked edge which generates new naked edge.

再帰的にコンターを作る

フィードバックループを利用し、ジオメトリをオーブンエッジから生成されたパイプによって切り取ることによって、再帰的にコンターカーブを生成します。

21-B. Marbling with solver

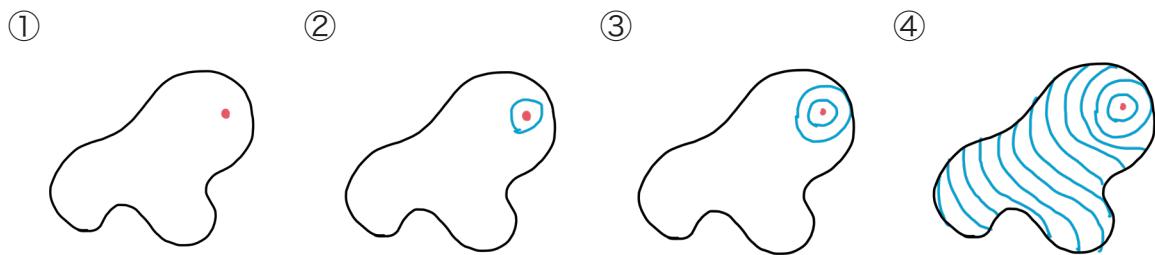
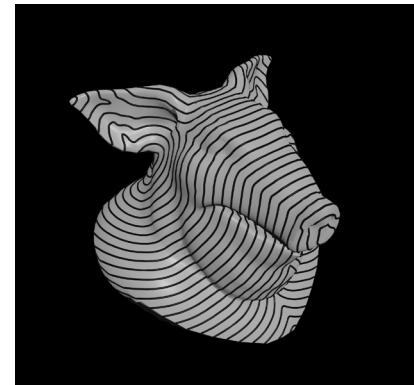
ソルバーを使ったマーブル模様



21-surface_marbling-B.hiplc

This example uses Solver instead of feedback loop which makes it possible to create generative animation just like fractal geometry. This example is pretty much the same with previous example just that it uses Solver SOP instead of For Each loop.

この例ではフィードバックループの代わりにSolverを使っています。それによりフラクタル形状を作るときと同じように、生成的なアニメーションを作ることができます。この例ではFor Eachの代わりにSolverを使う以外は、前の例とほぼ同じプロセスを踏んでいます。

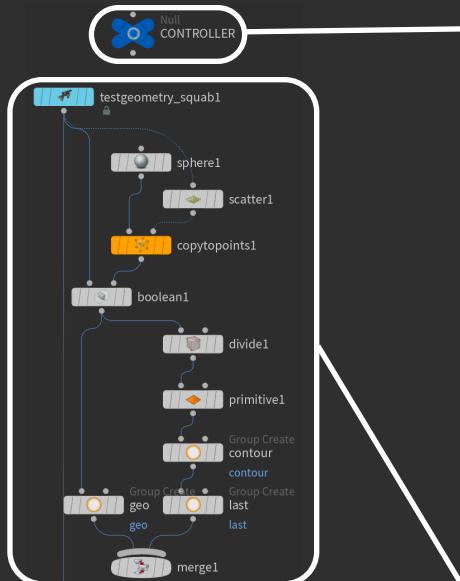


Create an initial point on a geometry.
ジオメトリの上に初期のポイントを作る。

Place sphere on a point and trim geometry with it to get naked edge.
ポイントに球体を配置し、それを使ってジオメトリを切り取り、オープンエッジを取得する。

Create a pipe shape from a naked edge and trim geometry with it to get new naked edge.
オープンエッジからパイプを作り、それを使ってジオメトリを切り取り、新たなオープンエッジを得る。

Continue Step 3 to create marble pattern on the geometry using Solver.
ソルバーを使ってステップ3を繰り返し行ない、ジオメトリの上にマーブル模様を作る。

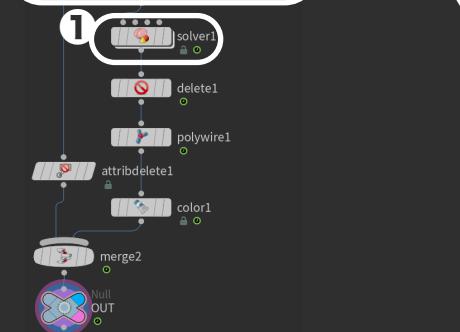


Setup parameters

- **thickness:** distance between contours
- **pt_num:** number of initial spheres
- **seed:** random seed for sphere position

パラメータの設定

- **thickness:** コンター間の距離
- **pt_num:** 初期球体の数
- **seed:** 初期球体の位置のランダムシード

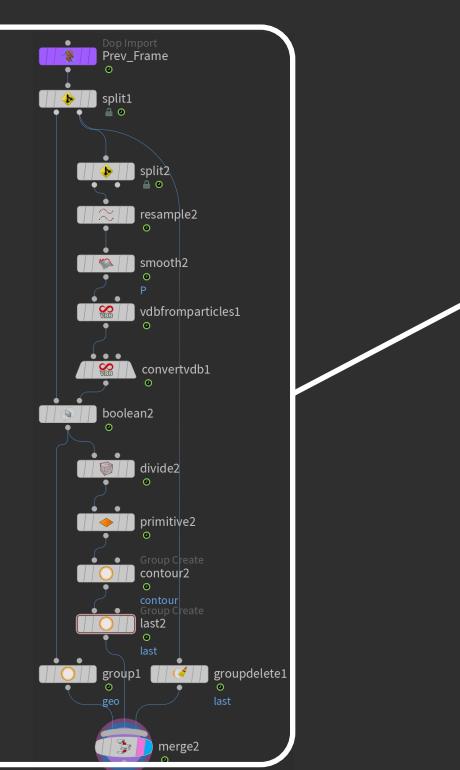


Get initial naked edges

Get initial naked edges by cutting geometry using number of spheres.

初期オープンエッジを作る

初期のオープンエッジを、ジオメトリを複数の球体で切り取ることで取り出します。



Recursively create contour

Use Solver SOP to recursively create contour curve on geometry by cutting geometry using pipe generated from last created naked edge which generates new naked edge.

再帰的にコンターを作る

Solver SOPを利用して、ジオメトリをオープンエッジから生成されたパイプによって切り取ることによって、再帰的にコンターカーブを生成します。

Conclusion おわりに

Thank you again for having an interest in this book. I hope there was something useful you gained from this book and I hope you enjoy exploring the unlimited number of possible solutions when you have to solve your own design problem.

本書に興味を持っていただきありがとうございました。本書を読んで何かしら得るものがあれば幸いです。そして自分自身のデザインのタスクを解く上で、無限にある解法の可能性を探る楽しさを感じていただけたらなによりも嬉しいです。

Other books 他の著書

Algorithmic Design with Houdini

Houdiniではじめる自然現象のデザイン (Japanese)

<https://www.amazon.co.jp/dp/4802511027>

Parametric Design with Grasshopper

建築/プロダクトのための、Grasshopperクックブック (Japanese)

<https://www.amazon.co.jp/dp/4802511213>

Nature Design with Houdini

自然界のアルゴリズムを利用したデザインレシピ集 (Japanese)

<https://orangejellies.booth.pm/items/1026131>

Neural Design with Unity and Houdini

強化学習とプロシージャル・モデルの組み合わせ手法集 (Japanese)

<https://orangejellies.booth.pm/items/1026122>

Tiling Design with Houdini

Houdiniを使った幾何学パラメトリックタイリング模様 (English & Japanese)

<https://gum.co/OVDgY>

About Author 著者紹介

Authour 著者

Junichiro Horikawa / 堀川淳一郎

Freelance algorithmic designer and architectural programmer based in Tokyo, Japan. Author of *Algorithmic Design with Houdini*, *Nature Design with Houdini*, *Neural Design with Unity and Houdini*, *Tiling Design with Houdini*, *Housing Layout with Houdini and Machine Learning*, and co-author of *Parametric Design with Grasshopper*.

Special Thanks スペシャルサンクス

Daiki Kanaoka / 金岡大輝

Digital fabrication master.

Algorithmic Design Workbook with Houdini

2020年3月1日 技術書典8版 v1.0.0

著者 堀川淳一郎
発行所 Orange Jellies

(c) 2020 Orange Jellies