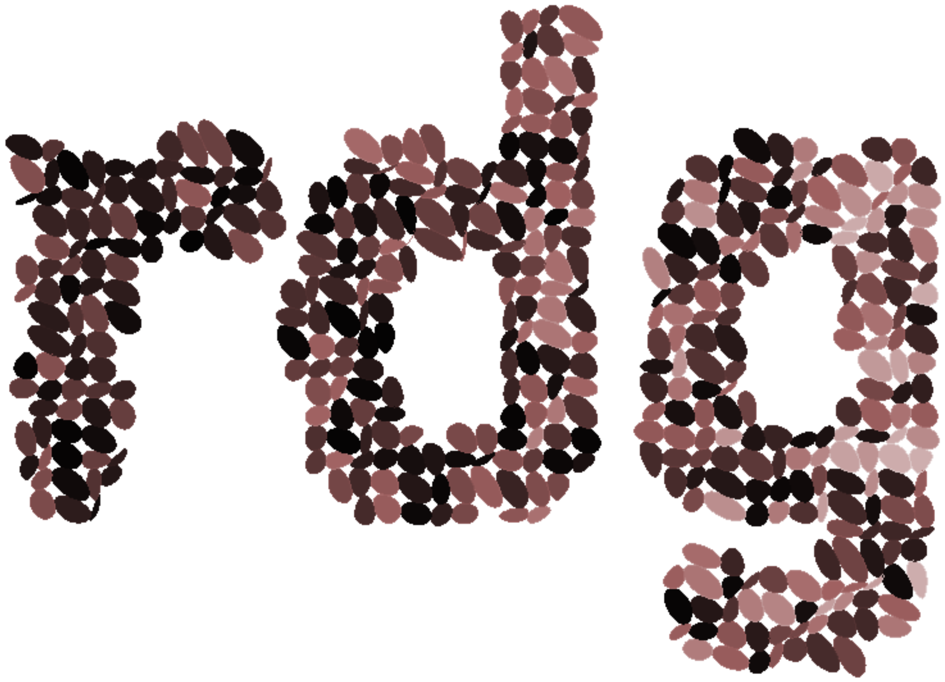
Poor Man's Circle Packing

—

Generating Procedural Geometry in Houdini

Georg Duemlein



The Industrial Parametrisation of the World
http://www.preset.de/
2008

Abstract

This document explains concepts and creation of procedural geometry in Houdini.
The example demonstrates how a template mesh can be used to create a complex structure.

Poor Man’s Circle Packing

Circle packing is an „arrangement of circles inside a given boundary such that no two overlap and some (or all) of them are mutually tangent“ [Weisstein]. There are several algorithms that allow to generate such arrangements [McNeel] and implementations for various languages like Java [McCullough]. Due to there iterative nature these algorithms cannot easily be implemented in SOPs.

Poor man’s circle packing is a „reverse engineering circle packing approach“. The underlying idea is to use a template mesh’s face edges to define the boundaries of each circle. Though resulting geometry isn’t a real circle pack the visual effect comes close enough. This technique allows us to create complex packs of non-uniform shapes which could only be achieved with great difficulty, if at all, using a brute force computational approach.

Circling the Squares

There are several imaginable ways to convert rectangular primitives to circle-like shapes. The results are of varying quality.

ConvertSOP - Circles

The ConvertSOP can convert faces to circles. These are circumscribed circles and of no great help for our task. Their geometry type is primitive and therefore they just inherit the orientation of their parent face without skinning themselves to possible deformations.

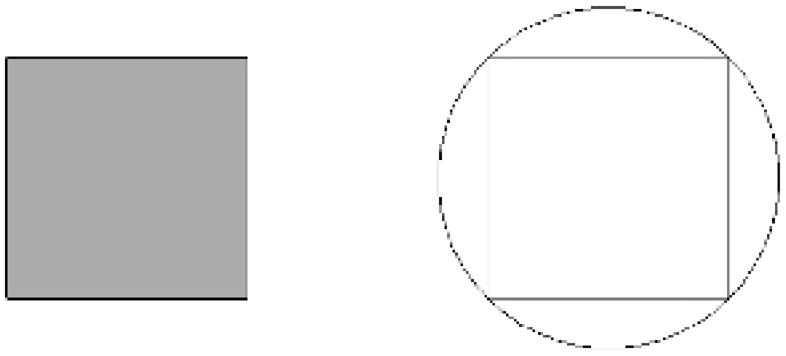


Fig. 1: A circumscribed circle created using the ConvertSOP

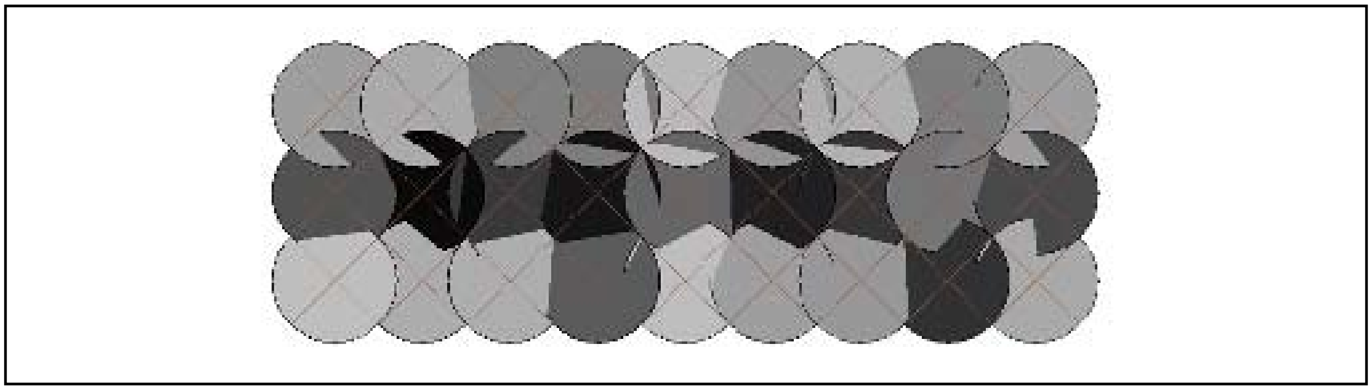


Fig. 2: ConvertSOP circles created out of a randomised grid

ConvertSOP – Nurbs Curve

This option creates a drop-like shape. Though this can be used to create trendy eye candy it doesn't create circle packs. The NURBs curves inherit the deformation of their parent face but it requires additional steps to forge them into real circles.

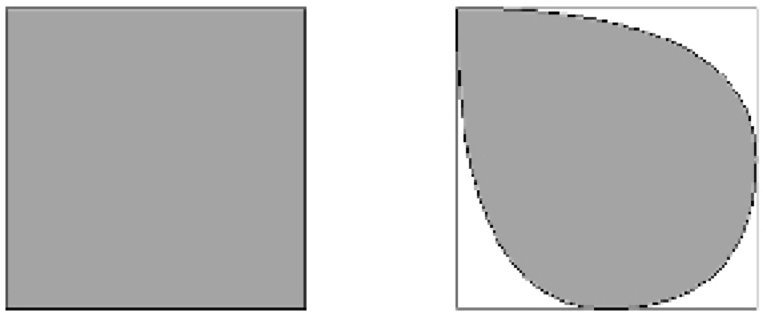


Fig. 3: ConvertSOP NURBS curves created out of a randomized grid

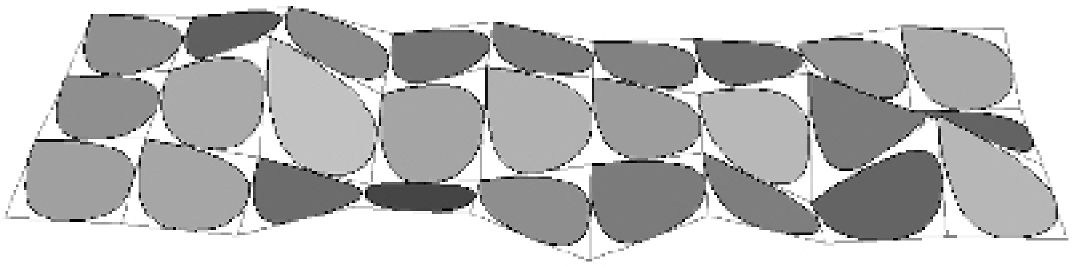


Fig. 4: A drop-pack created using the ConvertSOP

Rebuilding the primitive as a Curve

We could consider a face of the template geometry as a set of control vertices of a 3rd degree #NURBs curve. Four Cvs of a NURBSs Curve arranged as a square result in a circle like shape.
Due to the nature of NURBs it is not possible to create a circle using just four CVs also a rational Bezier spline is not capable of creating this shape [Wikipedia]. The NURBs circle needs to be scaled up to match the boundaries of the template face. The leaf-like shape resulting of a 3rd degree Bezier Curve results in nice patterns but not a circle pack.

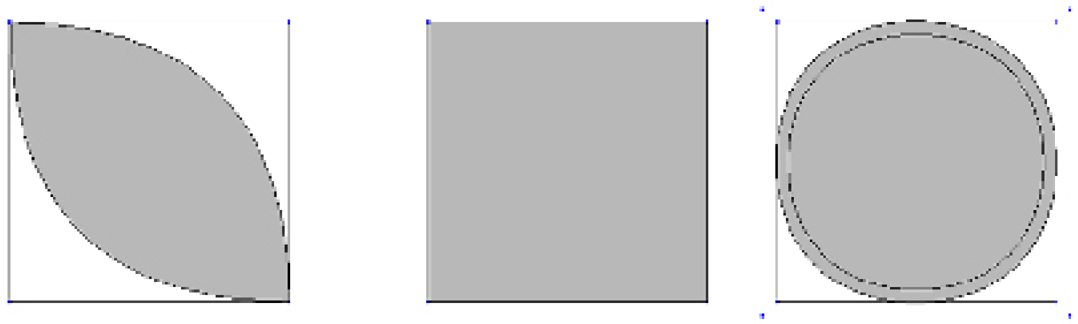


Fig. 5: Four CVs and different curve types: Polygon, NURBs and Bezier.

Procedural Creation of NURBs Curves

Usually we use the interactive mode of CurveSOP to create curves in the view port. This generates a string of coordinates of the points of the curve. The parameter supports a alternative syntax to reference a point positions of the input geometry:

p***n*** with n=0..($NPT – 1) of the input SOP.
Example: p0 p1 p2 p3

All we need to know is the point count of the template face. There are many ways to construct this string. My experiments showed that the following approach is useful:

1. A sortSOP sorts the points of the primitive by „Vertex Order" as the point numbers aren't necessarily in the right sequence.
2. We now group all points we are going to use as CVs with a groupSOP. It is important to enter *$OS* as group name. which causes the group to have the same name as the operator.
3. In the coordinate parameter of the curveSOP we enter this expression:
```
`"p" + strreplace(pointlist("../" + opinput(".", 0), opinput(".", 0)), " ", " p")`
```
4. done.

Let’s take a look at the expression:
opinput(*name, index*) returns the name of the SOP connected to a input of another SOP. In our case this is the name of the preceding groupSOP.
pointlist(*surface_node, group_name*) returns a space seperated list of point numbers.

As we named the group *$OS*, which is the name of the operator, *pointlist(“../” + opinput(“.”, 0), opinput(“.”, 0)* returns all point numbers grouped by the groupSOP: *0 1 2 3*

strreplace(*string, old, new*) replaces all occurrences of old with new in string. In our case all spaces in the point list will be replaced by „ p“ - efficiently adding a „p“ befor all point numbers: *0 p1 p2 p3*

All we need to do is to add a prefix „p“ in front of the string, as the first point number wasn’t affected by the strreplace function: *p0 p1 p2 p3*

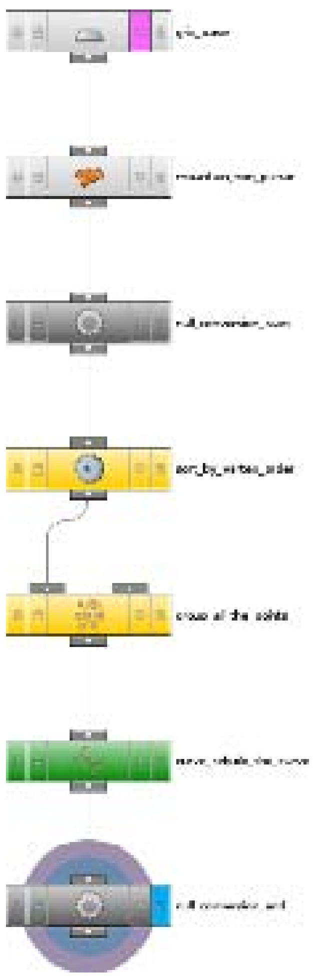


Fig. 6: The network to rebuild a arbitrary template face as NURBs curve

At first glance this approach might seems to bear a overhead as it is just creating a curve, but offers several advantages:

1. It works without user interaction and can easily be converted into a HDA.
2. Grouping the template points allows us the process just a subset of points in polygons having more than 4 points.
3. The expression syntax compared to a hard coded string like „p1 p2 p3 p4" in turn keeps the curveSOP independent of the point number of the input geometry.
4. The template geometry doesn't need to be constructed of planar regular quads or squares

Curves of different orders require a specific count of points: n >= order – 1.
Four points can build a 5th order curve which generates a smooth circle-like shape.
Due to the underlaying maths the area of the resulting shape is much smaller than the area of the template face. Therefore the circles don't touch each other.

To compensate this we add a primitveSOP to re-scale the circles.
It might be possible to calculate the scale factor by comparing the areas of template and circle face.
Trial-and-error lead to the following values:
Factor 1 for 3rd order, factor 1.09 for 4th order and 1.2 for 5th order curves.

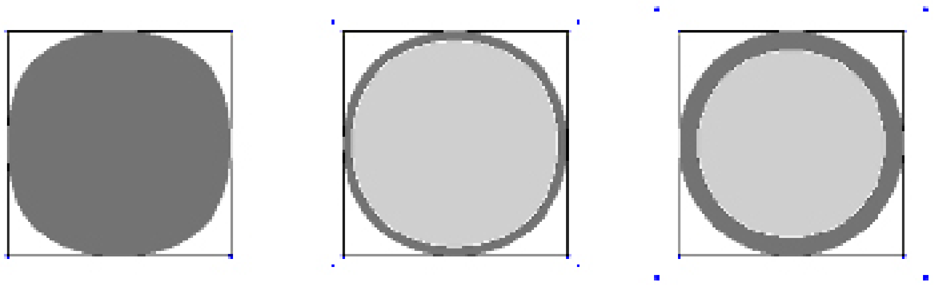


Fig. 8: Compensation of area shrink for different curve orders: 3rd, 4th, 5th

Rebuilding the template mesh

Creating a network that proceduraly converts a template mesh into circles is now a pretty straight forward task. We loop over all faces of the template geometry and convert each into a circle using the technique described.

First we need to make sure that the template geometry is a polygon mesh and in case of being a different geometry type we use a convertSOP.

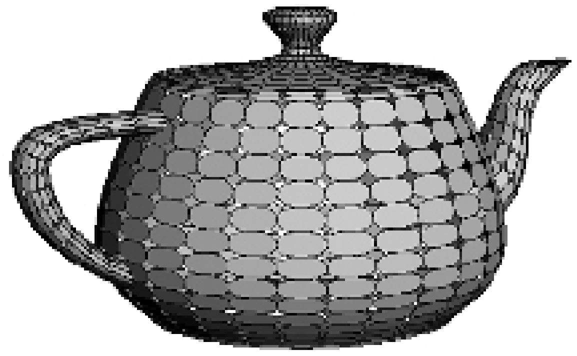


Fig. 9: Utah teapot converted to polygons and rebuilt as circles

This example demonstrates that the technique is a. scalable and b. doesn't rely on planar geometry.
Though it might be possible to copy shapes to face centres and deform them using a raySOP or creepSOP.

Such a conversion can create polygons with more than four points. It's up to you if and how you handle this situations. Easiest way would be a switchSOP that executes different branches of tessellations or curve orders depending on the point number.

The loop itself is the generic copy-stamp-delete setup.

1. A nullSOP named „null_conversion_start" acts like a separator in the network.
2. A copySOP at the end of the chain creates as much copies as there are faces in the template geometry.
   Number of Copies: *nprims(„../null_conversion_start")*
   A stamp parameter *„cnum"* stamps the number of the current copy *$CY* upstream.[1]
3. A deleteSOP right after the separating nullSOP deletes all non-selected primitives.
   The affected group parameter can use a list of group names or primitive/point numbers to select parts of the incomming geometry. We use the later one to select the face with the current copy number:
   Group: \`stamp(„../copy_collect", „cnum", 0)\`
   As the Group parameter is a string we need to wrap the expression into backticks.

1 Don't forget to enable „stamp parameters" in the stamping folder

Every operator listed between the deleteSOP in the beginning and the copySOP at the end of the „loop“ will be applied to the remaining face the delete operation.

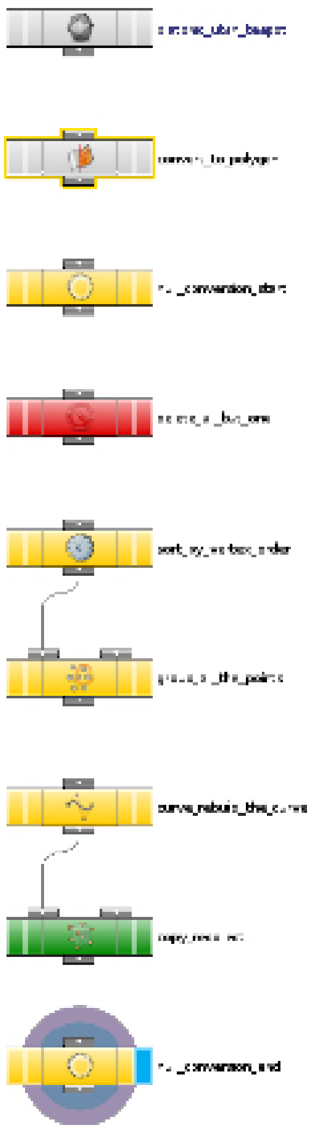


Fig. 10: The complete network

This is how every face of the template mesh is converted into a inscribing curve.

Conclusion

Though this approach lacks the visual characteristics of the traditional circle packing look [Stephenson] it can be used to create some complex looking patterns and structures.

The main difference is caused by the reverse engineering approach. Circle packing algorithms either contract a set of circles towards a arbitrary centre or fill a boundary shape with circles of varying radii.
Their aim is to pack as many circles into the area as possible. This creates the intrinsic feeling of natural or higher orderliness.

The poor man's circle packing approach uses a template mesh, therefore the look and arrangement of the pack is defined by the resolution and organisation of this structure.

Nevertheless the resulting curves can be further refined or used as a base for different geometry [Duemlein].

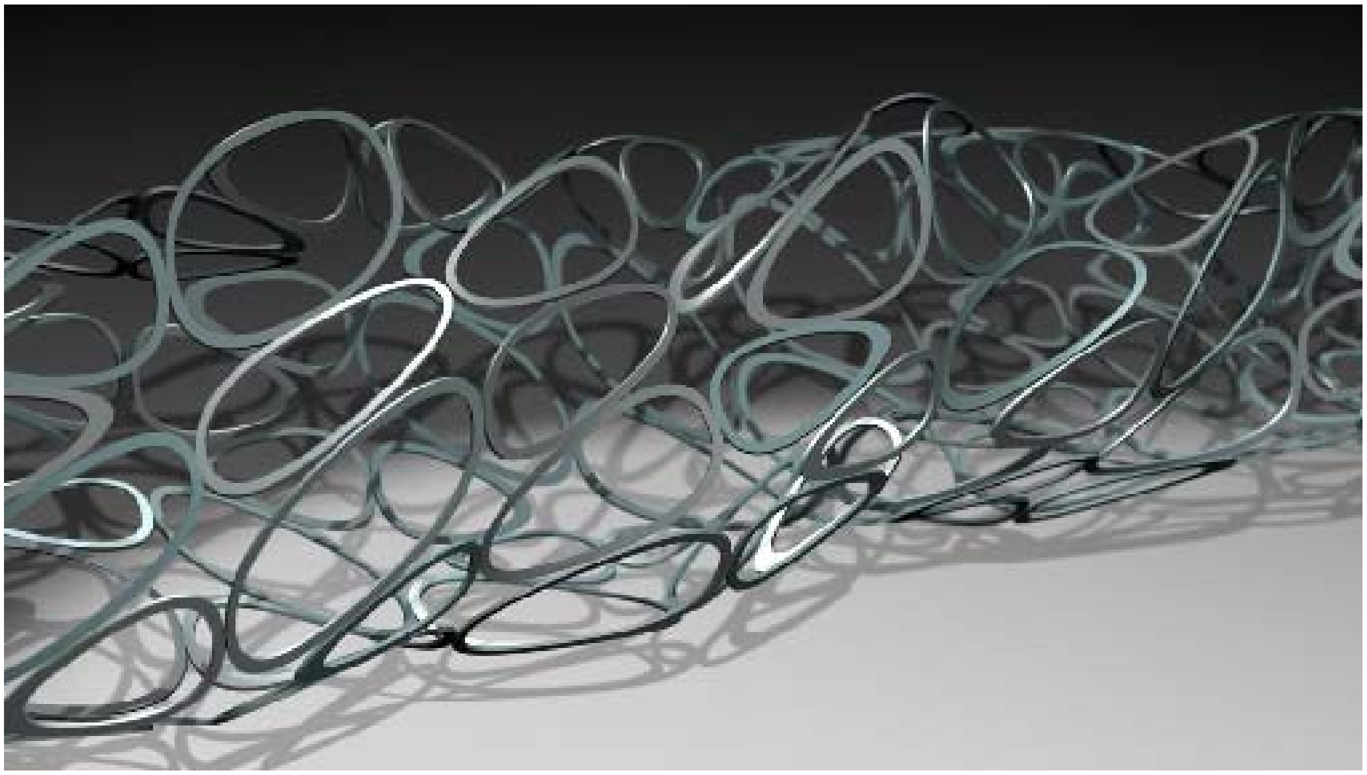


Fig. 10: A structure generated with the technique destribed in this document

February 2008
Georg Duemlein

References

Duemlein, Georg, pmcp:ontfm
http://www.vimeo.com/696476 [2/19/2008]

McCullough, Sean, Crickets Chirping, CirclePacking1 : Built with Processing
http://www.cricketschirping.com/processing/CirclePacking1/ [2/18/2008]

McNeel, Wiki, Circle Packing Plugin
http://en.wiki.mcneel.com/default.aspx/McNeel/2DCirclePacking [2/18/2008]

Stephenson, Neil, CirclePack and Circle Packing
http://www.math.utk.edu/~kens/ [2/19/2008]

Weisstein, Eric, MathWorld, Circle Packing
http://mathworld.wolfram.com/CirclePacking.html [2/18/2008]

Wikipedia, Nonuniform rational B-spline
http://en.wikipedia.org/wiki/NURBS [2/18/2008]