INF 574: Harmonic Coordinates for Character Articulation

Ayham Olleik and Guillaume Loranchet

December 2020

Abstract

This project aims at understanding and implementing methods presented in the paper Harmonic Coordinates for Character Articulation by Joshi et Al [1]. The paper presents a new method for cage deformation called Harmonic coordinates that allows a more local control and avoid negative values. In our project, we focus specifically on the 2D case.

1 Introduction

Cage deformation is mostly used in the case of characters articulation (or rigging). The main principle is to surround a complex model with a simpler cage that will be used to deform the model. We know that for each point v inside a closed surface mesh with vertices noted C_i, there exists some weight $w_i(v)$ such that $v = \sum_{i=0}^{n} w_i(v)C_i$. Then, if C_i' denotes the new cage vertex position, we can calculate the new position $v' = \sum_{i=0}^{n} w_i(v)C_i'$. The principle challenge is to compute weights that give the most accurate deformation. The main advantage of this method is that once the weight is determined, all the deformations have a linear cost with respect to the number of points.

2 Previous Work

The paper by Joshi et Al. compares their harmonic coordinates method with Mean Value Coordinate method presented by Floater et al. and later by Ju et al in 2005. Floater focused on a 2D method for quasi-convex polygons. However, it could only insure positivity inside the kernel of the polygon (set of points that can be attached to every vertex without crossing any edge).

Floater also worked on a method for 3D interpolation that we will not discuss here. In the 2D case, they compute w_i as follow: $w_i = \frac{tan(\alpha_{i-1}/2)+tan(\alpha_i/2)}{\|v-C_i\|}$. From this formula, we can see the two main disadvantage of mean value coordinates: non interior locality and negativity. Indeed, as it based on the Euclidean distance, a cage vertex could apply a relatively strong weight outside of its local view (for example if the point is situated in the opposite leg). The negativity is due to the fact that they take into account signed angles. This causes points to move in the opposite direction. The harmonic coordinates insure both interior locality and non-negativity.

Lipman et al (2007) developped the Green coordinates that, unlike mean value coordinates and harmonic coordinates, take into the normal values to preserve surface details. For example, if we move a cage vertex along the normal direction (e.g along the x-axis), then the normal direction should be updated with respect to the other axis.

3 Approach

Joshi et al. paper focused on character articulation thus the importance of interior locality and non-negativity that could be particularly visible due to the strong concavities of a character cage. The first issue they addressed if the usage of straight line distances that disregard the visibility of the model point from the cage vertex. Based on Floater et al. and Ju et al. methods, they defined a function f over the closed boundary and average over all points x in the boundary. However, instead of using Euclidean distance, they average over all Brownian paths leaving the point p inside the cage. The value of the path is equal to $f(x)$ with x being the first point on the boundary that the path intersect. Several papers like Bass et al. in 1995 showed that this problem was equivalent to solving the discrete Laplacian and building the harmonic coordinates at each cell. In our code, we take a 2D Model to describe and show a working model that can be extended to 3D dimensions with a trickier implementation.

4 Technical Implementation

4.1 Building the Grid

We start to build our input/model and the cage around it. Then we follow the suggestion of the paper. We first try to divide an area that covers the model and the cage we have into a square grid such that the total number of cells in this grid is equal to 2^s where s is 6 in the first benchmark we run. Algorithm 1 shows the code to build the grid that covers the Model and Cage we have using a $Cell$ structure that contains a $Label$ of the cell and the $list$ of harmonic coordinates associated with this cell. At first we label all our cells as $UNTYPED$, which means undiscovered and yet have no meaningful label. Then we go and label the Cells as $BOUNDARY$, $EXTERNAL$ and $INTERNAL$ based on their positions.

4.2 Dealing with Boundary cells

We start first by the $BOUNDARY$ ones and Algorithm 2 shows how we find $BOUNDARY$ cells, all cage vertices and the cells that edges of the cage cells pass through are considered boundaries. It is also necessary to mention that when we fill a boundary cell we fill its harmonic coordinates also since we know them. The harmonic coordinates consist of coordinates of points with respect to the cage vertices. Thus, a vertix c_i belonging the cage at index i has coordinates zeros everywhere expect the ith position. In addition, any point on an edge of the cage can be represented with 2 non-zero coordinates which are the coefficients of a linear interpolation between the 2 vertices of the edge α and $1 - \alpha$ where $0 < \alpha < 1$.

4.3 Dealing with Exterior cells

Next, we assume that the cells at the boundaries of the grid are exterior or boundaries since the model is enclosed inside the grid and the cage. The next step, described in Algorithm 3, is to find the exterior cell. We start from the four corners of the grid and for each cell, if it's not labelled yet, mark it as $EXTERIOR$ and propagates to its 4 neighbors. Otherwise, it means they are $BOUNDARY$ and we stop.

4.4 Dealing with Interior cells

The final step would be marking the remaining $UNTYPED$ cells as $INTERIOR$.

4.5 Forcing zero Laplacian & Propagation

We rely in this section on the idea that the discrete Laplacian at a certain cell at x,y can be approximated as h(x+1, y) + h(x-1, y) + h(x, y+1) + h(x, y-1) - 4*h(x,y). The value we are changing is at cell x,y and thus it is enough to set the h(x,y) = (h(x+1, y) + h(x-1, y) + h(x, y+1) + h(x, y-1))/4 and propagate over all the $INTERIOR$ cells (the function only propagates on the interior of the cage). We evaluate the max cell change at each Laplacian iteration and we stop when we achieve a max cell change below a certain threshold t. Now, we end up with harmonic coordinates at each cell that could be used to update the position of points that are inside these cells.

5 Results & Discussion

5.1 Metrics & Numbers

In order to study the metric of this model we run some benchmarks with different parameter values of grid exponent size s and the Laplacian threshold t. We present to you our results in table 1:

#Interior Cells			
	s = 6	s = 10	s = 12
t = 10^{-5}	8	627	2867
t = 10^{-10}	8	627	2867

time(s)			
	s = 6	s = 10	s = 12
t = 10^{-5}	0.009	0.140	1.832
t = 10^{-10}	0.008	0.403	6.707

#Laplacian Rounds			
	s = 6	s = 10	s = 12
t = 10^{-5}	12	388	1267
t = 10^{-10}	26	1170	4726

5.2 Small Discussion

These previous results show that for few interior cells and only in 2D case, this model can become memory intensive. Especially if dealing with sophisticated, high resolution 3D objects. In addition, with low number of cells, controlling the cage cells to make the Model behave in a specific way was not trivial or always achievable especially when dealing with continuous change and jumps over the cage vertices. In the next section, we present to the reader some of the problems we faced in our results with a small interpretation.

5.3 Possible Problems

5.3.1 Tractability

In many tries, flipping the key combinations that lead the cage to the same final state but with different order of steps and updates like the two key combinations 1,1,1,1,2,2,2,2 and 2,2,2,2,1,1,1,1 lead to different behaviors as show in the next figures.

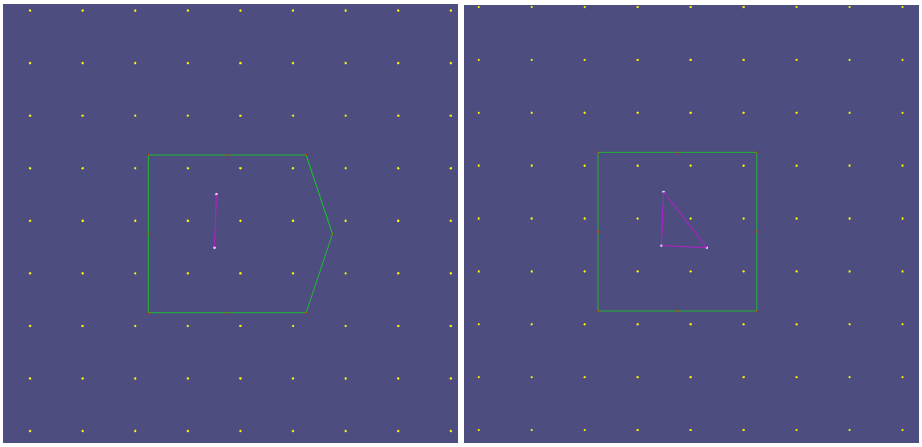

Figure 1: Left: 1-1-1-1-2-2-2-2, Right: flipped pattern

5.3.2 Exterior Jumps

In some other cased, any jump of an interior vertix of the model to one of the exterior cells where the function is not defined will push this vertex to the center of the viewer. That result is expected, and in order to deal with this, the harmonic coordinates need to be updated and the Lalplacian re-propagated whenever exterior cells become occupied by boundary or interior vertices. This problem then required repeating the whole process which will consume more time and resources. Next figure shows the result we mention, on the right image we see that the bottom left vertix is pushed to the center of the viewer. That is because the previous correspondent cell was exterior with zero undefined harmonic coordinates,

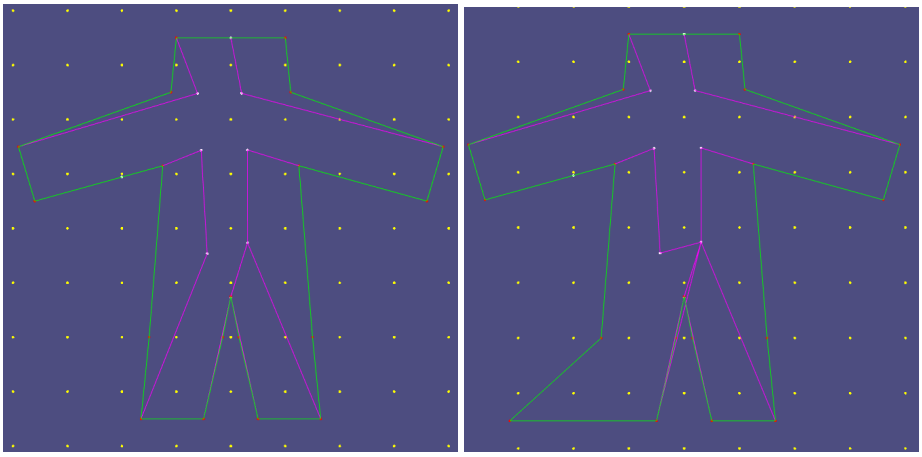

Figure 2: Left: original model, Right: exterior cell breached

5.3.3 Model Outside Cage

Another problem that could be faced in high end character articulation is the case where the cage bends the model in a way that the edges connecting the mode vertices cut through the exterior of the cage preventing the model from being totally contained inside the cage. Next figures (right) shows this issue. The right blue

edge is outside the cage.

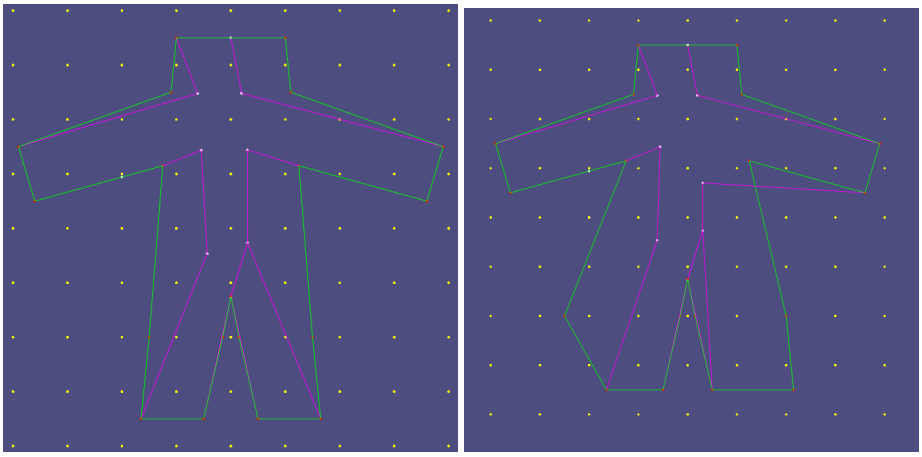

Figure 3: Left: original model, Right: model not contained

5.3.4 Precision & Smoothness

Increasing s and other parameters played a role in higher resolution and control over the model with the expense of memory and speed. Next few images present the model with different s values on a simple square model.

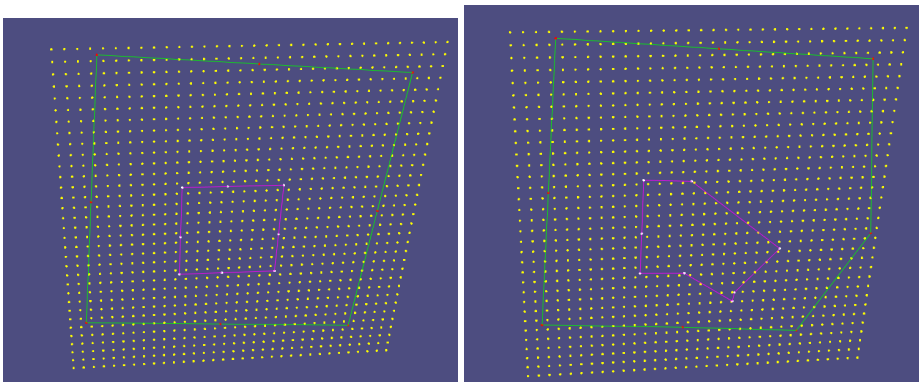

Figure 4: Left: original model s=10, Right: modified model

6 Conclusion

This problem remains subject to improvement in efficiency, speed and easy control over the point. It is also important to mention that the harmonic coordinate approach tackled and solved the problems of the previous methods like MVC [2]. For example, the harmonic coordinates retrieved by solving the Laplacian on inside did indeed prevent the negative weights and forced more smoothness and transition. Our code is available open source at this GitHub link with description on how to run the code.

References

[1] Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, Tom Sanocki. *Harmonic Coordinates for Character Articulation*. Pixar Animation Studios.
https://graphics.pixar.com/library/HarmonicCoordinatesB/paper.pdf

[2] Michael S Floater. *Mean Value Coordinates*.
https://www.mn.uio.no/math/english/people/aca/michaelf/papers/mean_value.pdf

[3] Mark Meyer, Tony DeRose, *Harmonic Coordinates*. Pixar Animation Studios.
http://graphics.pixar.com/library/HarmonicCoordinates/paper.pdf

[4] Jesus R. Nieto, Antonio Susin, *Cage Based deformations: survey*.
http://www.immagery.com/pdf/cage_based_def_methods.pdf

A Algorithms

Algorithm 1: Build Grid Vertices

function `Algo1` ($grid_{vertices}$, s, $offset_x$, $offset_y$, h) **:**

- $squareSideCells \leftarrow 2^{s/2}$
- $squareSideVertices \leftarrow squareSideCells + 1$
- $nbOfVertices \leftarrow squareSideVertices^2$
- $counter_x \leftarrow 0$
- $counter_y \leftarrow 0$
- $i \leftarrow 0$
- **while** $i < nbOfVertices$ **do**
  - **for** *j* **in** $nbOfVertices$ **do**
    - $x \leftarrow counter_x + offset_x$
    - $y \leftarrow counter_y + offset_y$
    - $grid_{vertices}.row(i) \leftarrow (x, y)$
    - $counter_x \leftarrow counter_x + h$
    - $i \leftarrow i + 1$
  - $counter_x \leftarrow 0$
  - $counter_y \leftarrow counter_y - h$
- **return** $grid_{vertices}$

Algorithm 2: Mark Boundary Cells

```
function Algo2 (grid, cageVertices, interpolationPrecision) :
    for v in cageVertices do
        (x, y) ← transformToCellCoordinates(v)
        grid[x][y].label ← BOUNDARY
        grid[x][y].harmonicCoordinates[i] ← 1
    α ← 1/interpolationPrecision
    for edge in cageEdges do
        (v_s, v_t) ← edgeVertices(edge)
        for j in interpolationPrecision do
            factor_1 ← j * α
            factor_2 ← 1 − j * α
            v ← factor_1 * v_s + factor_2 * v_t
            (x, y) ← transformToCellCoordinates(v)
            grid[x][y].label ← BOUNDARY
            grid[x][y].harmonicCoordinates[v_s.index] ← factor_1
            grid[x][y].harmonicCoordinates[v_t.index] ← factor_2
    return grid
```

Algorithm 3: Label outside cells

```
function explore_grid (grid,(i,j)) :
    if grid[i][j] == UNTYPED then
        grid[i][j] = EXTERIOR
        for p in grid[i][j].neighbours() do
            explore_grid (grid,p)
function exterior_labelling_init (grid) :
    n ← grid.size() − 1
    explore_grid (grid,(0,0))
    explore_grid (grid,(0,n))
    explore_grid (grid,(n,0))
    explore_grid (grid,(n,n))
    for p in grid do
        if p == UNTYPED then
            p ← INTERIOR
```

Algorithm 4: Propagate Laplacian

```
function Algo3(grid, laplacianPrecision):
    maxChange ← ∞
    while maxChange > laplacianPrecision do
        for v in grid_cells.INTERIOR do
            change ← updateToForceZeroLaplacian(v)
            if change > maxChange then
                maxChange ← change
    return
```