The importance of using low-level data in the parametric design process

A reflection and integration of the workshops preceding the graduation project

Reflection report by Roel Westrik

Tutors: Henriette Bier
Sina Mostafavi
Ferry Adema

Abstract

In the parametric and computational design process we deal with different types of data. Low-level data can be characterized by its proximity to base-level communication of the software and the processor, high-level data can be characterized by its many layers of computational mathematics and its geometric representation in the form of meshes or BREPs. During two workshops preceding the graduation project I found a common mistake of converging low-level data into high-level data too early in the design process. This resulted in designs which were both geometrically and esthetically not optimized. During my graduation project I had a different approach, where I would utilize low-level data throughout the entire project, to make sure both the design and the process were streamlined and integrated as a whole.

Keywords: architecture, base-level communication, computational design, data, design strategy, mathematics, parametric design, software

09/04/2019

Introduction

In my graduation project, the main goal was to design a pavilion in Vienna, based upon the music 'La Valse', written by Maurice Ravel in 1920. Dealing with such an abstract conversion of the arts, namely music and sound to geometry and matter, I chose to utilize a computational and parametric approach to ensure a consistent design approach. The main software used in this process is called Grasshopper, an extension to the widespread 3D modeling software Rhino by McNeel.

In the parametric design approach we are dealing with many types of data; from the inputs and parameters we find to the actual geometry we are dealing with. However in this reflection paper I want to focus on two types of data, the types which I call low and high level data. In the first part I want to define what I mean with 'low-, and high-level data'. Then I will reflect on the two workshops preceding the design project, and to conclude, how I incorporated the data types in my own graduation project.

Low-, and high-level data

In Grasshopper, we are dealing with visual programming. This means we are programming like any other text driven programming language, but we are almost always relying on a visual feedback. This visual feedback can be a mesh if we want a building, a BREP if we want a simple shape, or simply a set of numbers if we are performing an analysis. In other words, we are dealing with a lot of different types of data. All forms of data are calculated, as the software (Rhino, Grasshopper) communicates with your computer's processor. This communication is called base-level communication. The layers of data that lie closer to this base-level, I call 'low-level' data, also known as primitive data. When we think of low-level data, we think of Booleans (0's and 1's), characters, numbers, colors, point, lines and planes. As this data consists of merely simple numbers, all transformations are mathematical, and hence, quite speedy.

High-level data is characterized by the fact that they are constructed through processing low-level data, and they are built upon many layers of complex mathematical computations. They always have a clear visual representation. Think of BREPs (surfaces, solids), NURBS curves and meshes. Because of their complex mathematical nature, and their many layers of mathematical computations, transformations can be considered physical rather than mathematical. Transforming and intersecting these types of data with for example Boolean operations are therefore quite slow and take longer to compute.

Problem statement

Within a parametric design project, the goal consists always of high-level data. We want a visual feedback of all our analyses, typically in the form of a mesh or BREP. However we always start with low-level data, as our analysis is most likely built upon simple numbers. At some point in the project we make a conversion, which can happen gradually or quite abrupt. However in a well-thought-through strategy, this conversion happens consciously. Even though we can switch back and forth between the two types, we still have to be mindful of which type we use in every part of the process to ensure a holistic and integrated design process. In the latter portion of this reflection report I will discuss some mistakes and learning points from the workshops preceding the graduation project, and how I incorporated what I learned in my final graduation project.

Workshop reflections

The first workshop dealt with multimaterial 3D printing. Using our own code, we were able to hack the Ultimaker 3D printer to move in our own patterns, rather than in a singular extruding motion, which most 3D printers typically do. The challenge was the relation between the two materials, how did they meet and connect. The second workshop was held in Dessau, Germany, and had a similar goal. The objective was to design a system where overlapping beams could form a structural basis for an outdoor piece of furniture. The overlapping pieces were to be designed so that they could fit together and be assembled by a KUKA robot arm. Reflecting on the workshops, I found a common error: the timing and method of switching from low-level and high-level data was not optimal.

The first workshop dealt with the relationship between two 3d printed shapes. We started with a pre-generated shape, based upon existing geometry. A second material was generated top-down, and the intersections were handled with Boolean operations: the shape of material A was subtracted from material B so that they fit together nicely. However both materials consisted of high-level data, and the result was not a nicely integrated one (image 1 and 2). In retrospect I propose a different approach: we could have used a singular set of points, were, after analysis, we found points suitable for set A and B, then used voxels to generate a final design, where material A and B fit together nicely. In this way, all medial steps would consist of low-level data and the final design would more coherent.

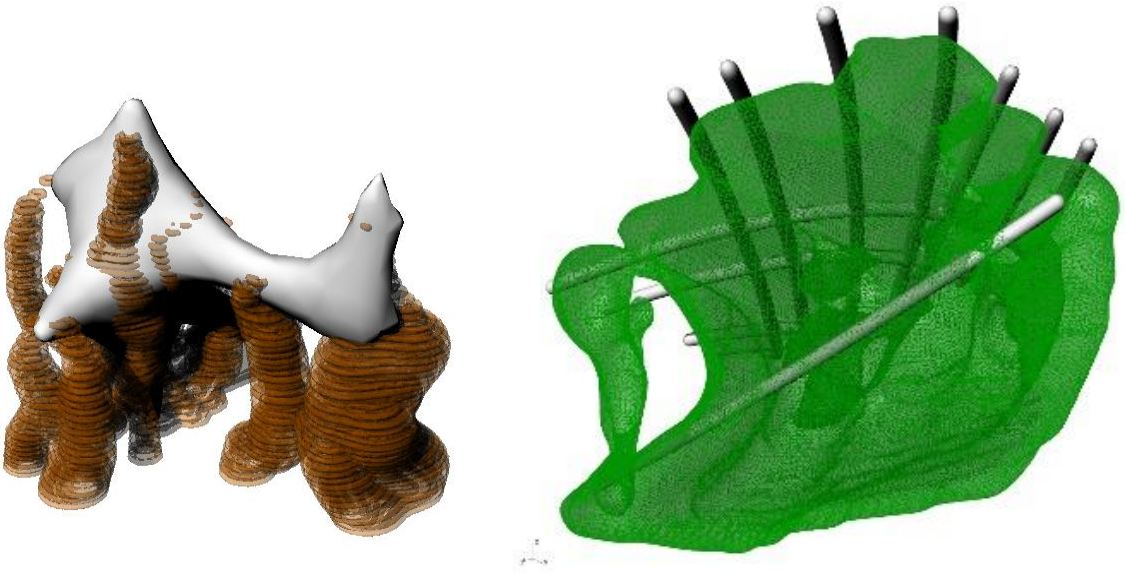


Image 1 and 2: results from workshop 1, with material A (grey) and B (red, green respectively)

The second workshop dealt with the intersections of pairs of beams. We used a method where half of each shapes of the beams would be subtracted from the other half of the pair at each intersection point using, again, Boolean operations. This approach has similar problems to the ones in workshop one (image 3 & 4). I would propose an approach where the beams are generated from the nodes, rather than the nodes being generated from every beam.

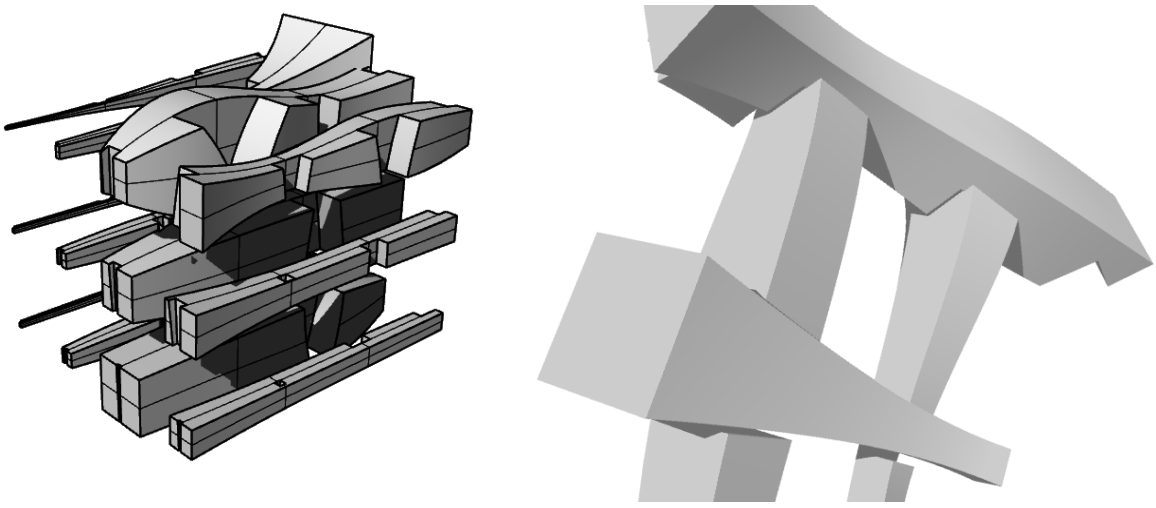
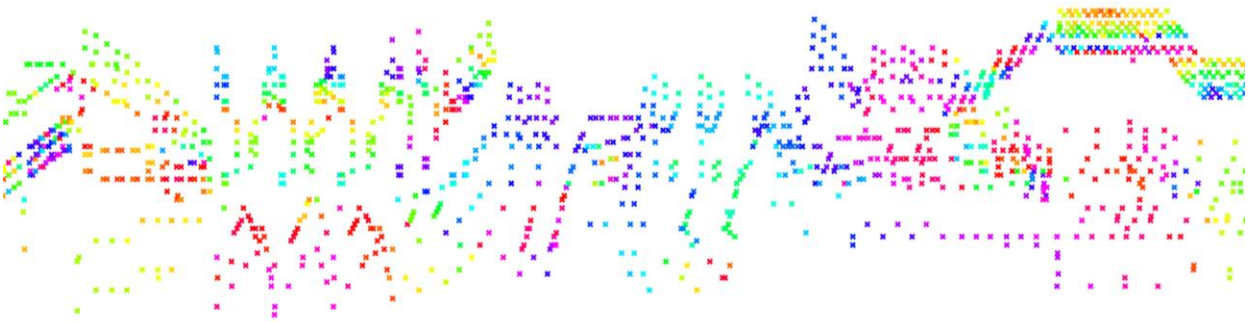


Image 3 and 4: results from workshop 2, constructive elements intersecting

Graduation project approach

During the graduation project, I faced challenges similar to those in the workshops, but with the new knowledge, I could handle them in a different approach. I will discuss three of my strategies involving low-level data as a result of the new design strategy.

Even in the early design stages I faced a difficult task: finding a strategy where I could translate music to a visual representation in a tangible and comprehensible way. This was an important step as this formed the first building blocks for the entire project to be built upon. My approach was to convert the entire piece of music to color. Every single note would be assigned a singular color value, based on its pitch, volume and relationship to its neighboring notes. Three of these aspects would be stored into a single color, used aHSL color space. As there were 12000 notes in the particular piece of music, I averaged many colors of roughly the same aHSL value, reducing the entire composition to 91 colors, each has internally stored the average harmonic center, complexity and loudness of the music. The gain access to the colors I could therefore use simple mathematical interpolation, rather than storing all data in a complex matrix of values and accessing and processing each value individually. In other words: I was able to get a visual representation of an otherwise invisible data set, using low-level points and colors, rather than a complex, high-level, geometric strategy.



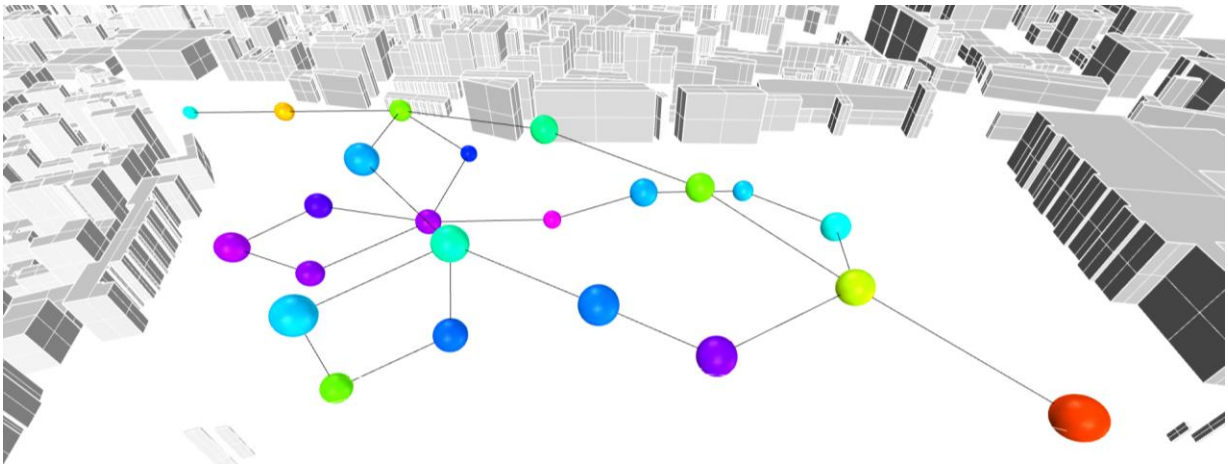


Image 5 & 6: incorporation of low-level data (colors) to generate an architectural diagram describing circulation and volume distribution

After generating the diagram for circulation and volume distribution, using the above described coloring method, I was eager to get a visual feedback of my design in a more tangible way, but rather than wrapping the geometry with a high-level mesh in such an early stage, I used a vector field, which merely outputs iteratively generated points and lines, which keep the data at a low level (image 7). Generating a mesh at such an early stage would not only mean that whatever was generated was most likely messy, because of the lack of preceding low-level data, but would also mean that we would be stuck with working with the same mesh for the continuation of the project. Using a vector field to generate a pseudo-geometry as a visual feedback, would ensure that we keep the data at a low-level so that any consequent analyses would be meaningful and easy to process.

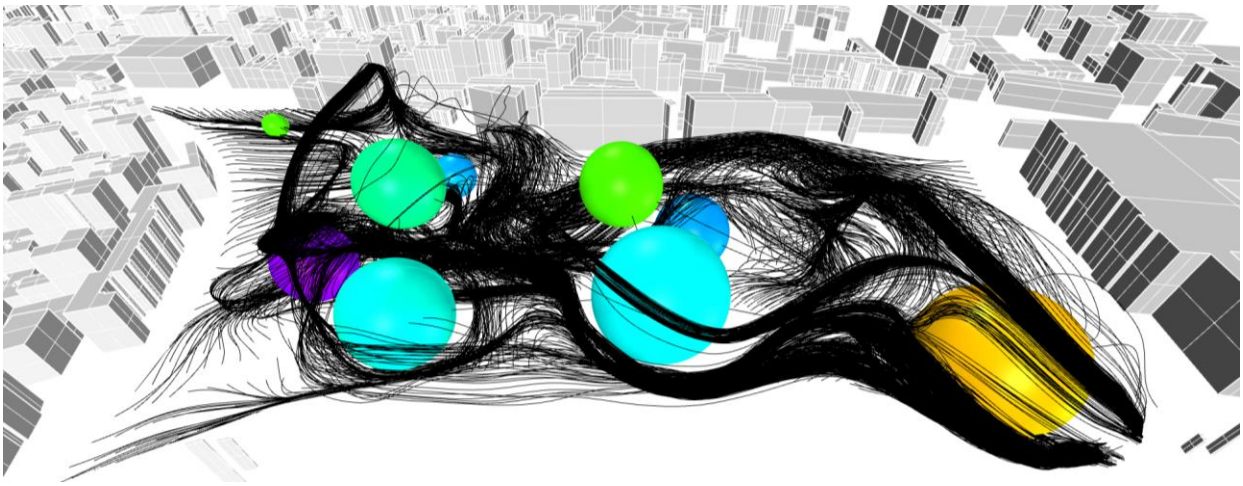


Image 7: using a vector field as tool for analysis (left), rather than prematurely meshing the dataset (right).

After the output of the vector field, I was again challenged not to immediately jump to a high-level mesh. I was faced with a similar problem as workshop 1: dealing with multimateriality. While I was at first tempted to generate an entire mesh from the group of output curves, after which I would analyze this mesh to divide it into structural and expressive elements to divide the materials (similar to the approach of workshop 1), I chose to keep working with the low-level data of the group of points and

lines. After a rudimentary structural analysis I was able to divide the curves into ‘structural’ curves and ‘expressive’ curves. All geometry that was eventually generated, were generated from the same, low-level, group of curves. Therefore the geometry is always connected, not only physically but also, more importantly, expressively. All layers of analysis are performed on low level sets of data, therefore the resulting geometry is always a final result only, and does require more further complex physical alterations.

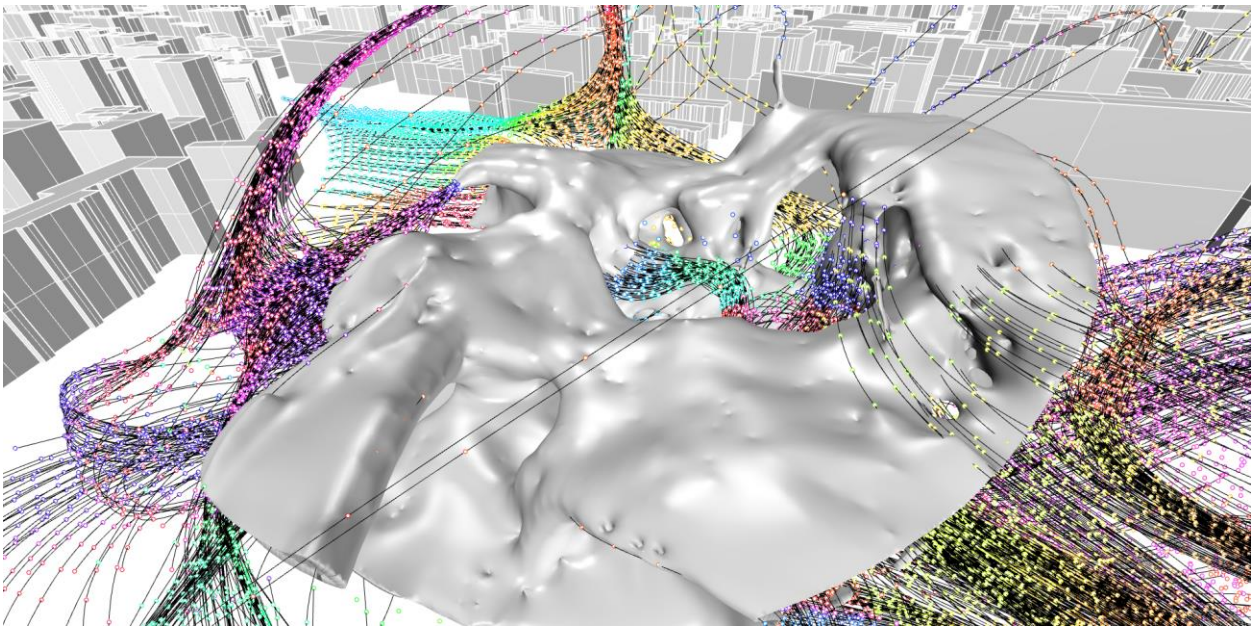


Image 8: Generating structural and expressive elements from the same data set

Conclusion & reflection

In a design process we are inevitably dealing with both low-level data, as well as high-level complex geometry. My advice (especially to new students in the studio, who are most likely new to the parametric design process) is to work with low level data as much as possible. Not only will this ensure a streamlined design strategy, where every step of the way is easy to compute and process, but this will also lead to a holistic design approach and a fully integrated design. One might be tempted to jump to conclusions and generate high-level geometry in an earlier design stage, but it is difficult to make the switch back to low level after this. If one requires tangible visual feedback in the form of, for example, a closed mesh or untrimmed surface, it might be best to generate these at a low resolution, then continue the rest of the project using its primitive data. Keep subdivisions or scaling factors for the last stages of the design project. Physical intersections and Boolean operations should never be required as the base of analysis in a streamlined process.

The greatest difficulty in using this design approach in my own graduation project was the never ending lack of visual feedback of closed solids. A set list of colors means nothing if you don’t see eye to eye on what it represents. This means discussing ideas and visions with tutors was a challenge, as it is difficult for different people to perceive the same meaning from an abstract set of data. However as more and more meaning is unraveled, more and more concrete ideas will materialize from the primitive layers of data.