Applying Informatics Knowledge to Create 3D Worlds

Michael Weigend

Westfälische Wilhelms-Universität Münster, Fliednerstr. 21, 48149 Münster, Germany
michael.weigend@uni-muenster.de

Abstract. Designing three-dimensional models using a tool like Google SketchUp is an attractive and inspiring activity fostering spatial thinking and visual creativity. The basic functions of SketchUp are easy to learn (low threshold). But more demanding design projects require computational thinking. This paper discusses some informatics concepts 3D-desigers need to know to be able to use SketchUp efficiently.

Keywords: Algorithms, Computer Science, Modeling, Thinking, Didactics

1. Introduction

Create the house of your dreams. Reconstruct a historic building that has been destroyed using verbal descriptions, old photos and plans. Design a city to be built on Mars.

Modeling three-dimensional worlds can be a thrilling and inspiring activity. Google SketchUp is an easy-to-learn modeling tool which enables even young students to construct complex surface-oriented 3D-models in a relatively short time. What makes the usage of SketchUp easy and efficient?

Perhaps, the secret of success is that you can use already existing everyday knowledge. Fischbein [4] and diSessa [3] use the term “intuition” for mental concepts, people feel certain about and which they have successfully used in many different domains (see also [13]. Sometimes intuitive concepts are subconscious, but still they influence our thinking and acting. For example, we all know what it means to move a thing, to pull a rubber band or to fold a sheet of paper. This is intuitive to us. Powerful and efficient geometric operations that can be performed with SketchUp rely on this kind of knowledge. But obviously specific concepts from the domain of geometry are helpful too. At least for some design tasks users need to know what points, circles, angles, tangents or irregular triangulated networks (TIN) are. What about informatics knowledge? Wing describes “computational thinking” as a “universally applicable attitude and skill set” that everybody should learn at school [14, p. 32]. Since it is claimed to be universally applicable, the question arises in what way computational thinking might be

helpful for creating 3D worlds. This is the quest this paper deals with. Relevant concepts, which I am going to discuss include the notion of objects, being in a state, classes, aggregates, parameters of method calls and iterations over data collections. Such knowledge can be learned separately in computer science lessons or "just in time" during the work with SketchUp. But however, they should not remain in the dark. They should be explicated and 3D-designers should be aware of them. A similar discussion takes place in the context of other digital artifacts like word processors [1, 11].

2. Values and Mutable Objects Being in a State – Mathematical and Informatical Perspectives on Geometry

Fundamental concepts in geometrical modeling are lines and points. In geometry, a line is defined by its properties, for example the position of the two end points. If you change one of these, you have a different line - a different entity. Thus lines are immutable. Nevertheless, in a context like constructing a two-dimensional geometrical shape with pencil and paper, people use phrases like "I will make this line a bit longer", thus indicating that they consider a line as a quasi-material entity. Its length is just a momentary state that can be changed. Furthermore, while designing geometry, people use colors and create blue and red lines. Of course the color is just decoration to make the geometry easier to perceive. When we make the distinction between the mathematical entity "line" and the visual entity that can be seen on the paper or the blackboard, we are entering the domain of informatics. The line segment we draw using paper an pencil or a software tool is an object, not *being* line segment but *representing* it in its state. Usually this object has additional – non-mathematical- properties like color, thickness, texture.

In dynamic geometry environments (DGE) like GeoGebra, GEONExT, and Geometer's Sketchpad students can construct geometric shapes like quadrilaterals on the screen with the mouse [2, 8]. The user can drag points and watch the changes of dependent lines and points, thus discovering geometrical knowledge. Keith Jones' analysis of math lessons with a DGE suggests that the notion of geometric figures being objects with changeable states is quite intriguing for teachers and students. For example, 12-years-old Karol said: "Our old math teacher used to call a rhombus a drunken square, because it's like a square but sick." [7, p.74]. Hölzl points out that students using DGSs must be aware of the behavior of the objects they manipulate on the screen. "And some of those characteristics may not at all lie in the realm of geometry" [6, p.171]. It might be helpful to distinguish clearly between the *view* on the screen representing a geometrical figure and the conceptual *model* underlying it. A similar distinction has been made in the "model-view-controller" pattern [10], which in software engineering is commonly used for the development of visual programs. For example, the model behind the view of a quadrilateral on the construction board is an aggregate of linked objects, which can execute certain operations like changing position. The object representing a line segment stores in its attributes references

to its two end points. When we drag one of these point-objects, we change its state but not its identity. Thus this point is still the end point of the line segment it is connected to. In consequence the line segment looks different now although we did not execute an operation on it. When a student drags the four points of a quadrilateral to the same place, it looks like one point. In mathematical reasoning (which is intended when using a DGE) the student actually thinks of just *one* point (view). But on the model level there are still four points, which can easily be separated.

3. Creating and Using Components: Class and Instance

From DGEs back to 3D-modeling with SketchUp, which is not an educational tool but a professional editor used for construction. In complex design projects it is convenient to use components. A component defines a smaller 3D-entity which is a part of the whole model. The model of a table can be constructed as an aggregate consisting of one instance of a slab-component and four instances of a leg-component. Like a class in OOP a component can be considered as a template (representing a type of entities), which can be used to create instances. All the visual properties you define while creating a component can be regarded as class attributes. These attributes have the same values in all its instances. If you change a component, all instances will adopt the changes. Thus a new instance of a component, representing a *type* of entity, is something different then a copy of an existing entity. There are additional instance attributes (equivalent to object attributes in OOP) that belong to the individual instances of a component. These include the scale, visibility on screen and the material the surface looks like. There exist things in real life, which are similar to templates. For example, a potato stamp determinates how the printed entity looks like. But – in contrast to SketchUp-components – real life templates must be defined completely before creating instances. If you change a stamp after having printed with it, the modifications have no effect on already created entities. When you build several houses using one common plan and change the plan later, the houses still look the same. Such templates in real life are used to *create* and not to define. The set of *created* instances is not an Aristotelian extension of a category. But when you edit a SketchUp component, you observe on the screen that all instances are in a magical way connected to the component and change simultaneously. You can see that small changes can have a large effect on the world and you realize that a SketchUp component actually defines a set of entities. The tables in Figure 1 are aggregates consisting of one slab and four legs each. Changing the - one and only - leg-component affects the whole picture. Even the shadows look different.

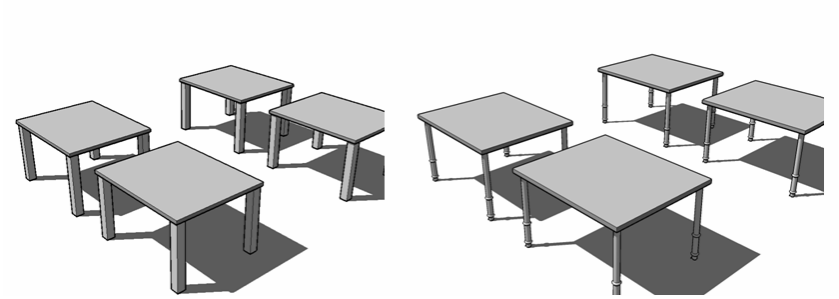

Fig 1 Effects of editing a component representing a table leg

4. How to put a plate on the table – different ways to define parameters

We are familiar with moving objects in our 3D reality. When Tom puts a plate on a table, he just takes it and moves it until it looks right. In the real world moving is a continuous process. This intuitive way to move things in space does not work in a 3D-modeling environment like SketchUp, since the user interface is two-dimensional.

In a 3D-modelling environment movement is one discrete operation. Consider a model consisting of instances of two components, one representing a plate and the other representing a table. To put the plate on the table Tom has to do three things: (1) select the move-tool, (2) specify the entity that is to be moved (plate-component) and (3) specify an Euclidian vector, which defines the shift (direction and length). To specify the entity and the operation (method) is easy. It is done by clicking on visual elements. The problem is how to define the vector. A quite elaborate way is this: Tom edits the plate-component and draws one or two lines underneath the plate to gain a point, which is exactly in the middle. Then he draws lines on the table top to indicate the target position of the plate. Note that drawing on an existing surface is easy in SketchUp. When you have selected the pencil tool and move the cursor over a surface, the system assumes that you want to put a line on it. With these two points on the bottom of the plate and on the table top Tom has (indirectly) defined the spatial vector for the move-operation. Since both points must be visible Tom changes the face-style to transparent. To execute the move-operation he selects the move-tool, clicks on the first point and then on the second on the table. The plate had followed the cursor movement and snapped to the target point after the second click. Finally Tom removes the lines he had used for the specification of the initial and terminal points of the vector. The beauty of this proceeding is that the parameter of the move-operation – the vector – is explicitly embedded in a meaningful context. The initial point of the vector is not just some triple of numbers but the center of the plate bottom. This relation to the

plate defines the meaning of this point. In the same sense the second point on the table top is a meaningful entity. It indicates the future position of the plate which is on the table and not too close to an edge.

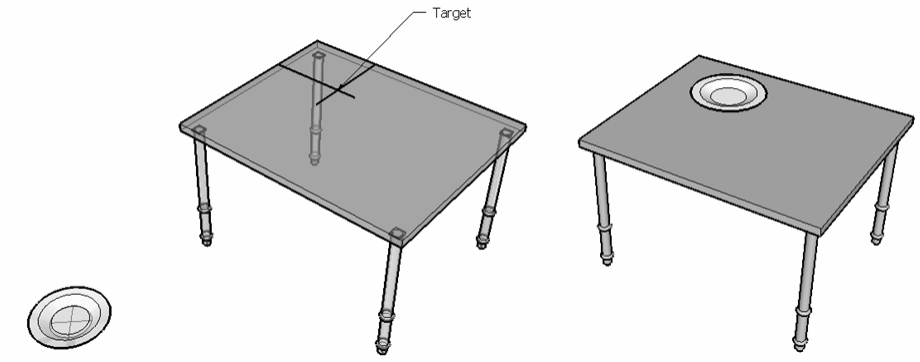

Fig. 2 Moving a geometry using a target point

Where is the connection to "computational thinking"? Computer programming includes embedding new knowledge in a context of existing knowledge, introducing new entities and constructing meaningful relationships between entities. Consider this Python program:

```
sequence = ["a", "b", "c"]
for index in range(length(sequence)):
          print index, sequence[index]
```

The `term range(length(sequence))` represents a list of numbers `[0, 1, 2]`. Instead of using explicit numbers, the programmer defines a list indirectly, using the already existing sequence. The term `range(length(sequence))` has a *meaning* within the algorithmic context of the program, but the list [0, 1, 2] has not.

A second way to put the plate on the table is to split the three-dimensional move into three moves along the three axes of the coordinate system. The first move could be in the direction of the vertical axis up to the height of the table. For this up-move-operation a vector is needed too. But it is specified in a different way – by its direction and length. Tom selects the move-tool, clicks on the plate and moves it in upwards-direction. But what does it mean to "move the plate"? When Tom moves the cursor, the plate is displayed at the current position of the cursor. When he strikes the escape-key, the plate jumps back to its original position. Only when he clicks the mouse, the plate remains at this position and stays there. In this moment – and not earlier - the atomic move-operation has been executed. The movements between the two mouse-clicks are anticipations of the future position of the to-be-moved entity. Generally speaking, Tom is specifying the parameter of an atomic move-operation by anticipating the future state of an entity. In the SketchUp environment his concept is additionally supported by a mechanism called inference. By touching the table top with the cursor Tom can specify a reference point. When he moves the plate along the vertical axis, and the

present height happens to be the height of the table, a dotted line appears connecting the plate to the reference point (Fig. 3). Again, this means the future position of the plate has been specified within a meaningful context.

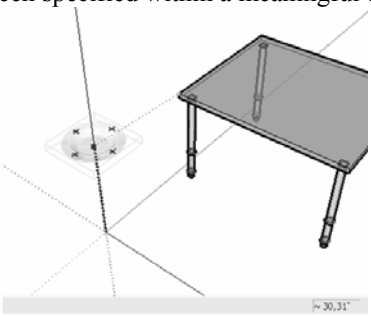

Fig. 3 Moving a plate-instance to the correct height using inference

Note that Toms did not use explicit data as parameters for the move operation. Of cause the system inferred the data from Tom's activity, but Tom himself used the concept of immaterial states. There still is another way to specify parameters: If Tom knows the exact height of the table, he can also enter the length of the move-vector in a small input field on the right side at the bottom of the window (Fig. 3). When he strikes the enter-key, the plate jumps to the corresponding position. This feature is called "accuracy" in the SketchUp world.

Let me sum up. I have described different ways to move an entity in a SketchUp model using concepts like anticipating a future state, embedding an entity in a meaningful context and inputting explicit data. When you see the move-operation through the eyes of a computer scientist, you are aware that this operation needs a spatial vector as parameter. The question is just how to tell the system the required data. Knowing a variety of activities, which all serve the same purpose – to specify a vector –, should make it easier to find an appropriate and efficient proceeding to perform the move in a given situation.

5. Iterations

In an iteration a certain activity is applied to each item of a collection like a set or sequence. In contrast to a while-statement, an iteration is not controlled by a condition but by a collection like a set or a sequence. The Latin word "iter" means "march" or "walk". Metaphorically speaking, to execute an iteration is to walk through the items of a collection and do something with *each* of them. Collections are specifically designed for iterations. This aspect is emphasized in the syntax of the programming language Ruby. Collection objects have the method `each`, to specify some activity to be applied to each item (example see below).

In 3D modeling with SketchUp we use iterations when we want to apply an operation on several geometrical entities. Assume that Sarah intends to paint several parts of a model in the same color. First she specifies a collection of

entities – using the selection tool – and then applies the material tool (paint bucket) on this collection. Note, that the selection tool is very efficient for defining collections. Sarah may create a "box" with the mouse, thus selecting all entities within the box. To exclude some of them she keeps the shift key pressed and clicks on those entities she does not want to paint. All selected entities are framed with yellow lines and are therefore easy to recognize as members of the collection.

The paint iteration can also be specified by a Ruby-program, which is to enter into the command line of the Ruby-console:

```
s = Sketchup.active_model.selection()
s.each {|entity| entity.material="blue"}
```

6. Attributes of entities

Suppose you have an instance of a component consisting of two instances of another component representing a cubicle (Fig. 4). How to put some material (like wood, metal or color) on the surfaces of this geometry? It can be done this way: Select the material tool (paint bucket), select a material (for example wood), move the bucket onto the collection and click. The second picture of Fig. 4 shows the result.

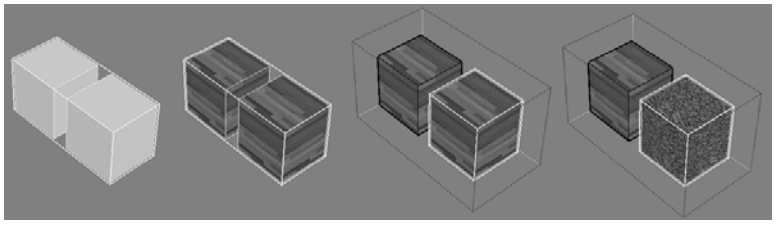

Fig. 4 Adding material to a Sketchup geometry

Now open the component and select one of the cubicles (third picture). Try to remove the material. This is not possible! But you can add a material to this entity, say a carpet, so that this cubicle looks different from the other (fourth picture).

This strange behavior can quite easily be understood by applying the concept of objects and attributes. The component instance representing two bricks is an aggregate consisting of two instances of a cubicle component (see UML object diagrams in Fig. 5). Each geometrical entity has an attribute named `material` defining the texture, shown on its surfaces. If this attribute has the value `nil`, the system uses the material specified in the aggregate-object for rendering. Thus the cubicle in our example, which is part of an aggregate and has no explicitly specified material, is shown with the material of the two-cubicle-component. And there is no way to change this by editing the properties of this individual cubicle.

But it is possible to add a material to this cubicle and make it look different from the rest of the aggregate.

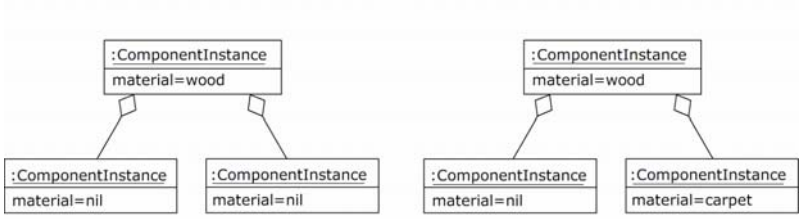

Fig. 5 UML object diagrams corresponding to the 2nd and 4th geometry of fig. 4

Note that this kind of construction does not happen in real life, where things are really *made of* material. In 3D-modelling the term material is just a metaphor for a property of a surface. Thus modeling using materials cannot be understood in a naïve way merely based on everyday experience. Formal informatics knowledge is required.

7. Conclusions and Pedagogical Perspectives

In this section I start with summing up the advantages of informatics knowledge for 3D modeling and then present a few ideas for teaching this knowledge.

SketchUp users should profit from informatics knowledge in several ways. Being aware of the underlying concepts increases the competence to communicate. Design plans and SketchUp system behavior are easier to understand and to explain to someone else, if you use proper informatics terminology (including terms like aggregate, object, attribute, operation, parameter etc.). This concerns teachers, supporting and scaffolding students working on 3D projects as well as students cooperating in a team.

Creating a complex geometry with SketchUp requires a deep understanding of model structures and the ability to combine operations to efficient design plans. If you do not know how to use iterations, collections and aggregates you are not able to create complex structures, just because you do not have enough time. In many cases it is necessary to construct auxiliary structures. For example, if you want to create a curved pipe, it is efficient to start with creating a box, which has nothing to do with a pipe, if you only consider geometry aspects. But if you have this box, you can easily create a curved line on the front faces of the box, then a circle on a side face, and then create the pipe using the follow-me tool and finally delete the delete the edges of the box. Such strategies are similar to the construction of class structures in OOP in order to manage cognitive complexity. 3D designers need to be creative not only in the domain of geometry and aesthetic but also in the field of informatics inventing efficient proceedings to implement their ideas.

Let us now consider the teaching perspective. Why should SketchUp be taught in informatics classes? Keep in mind that informatics concepts underlying 3D-

modeling exist independent from 3D-modeling and might be taught respectively learned more efficiently in a completely different context. Perhaps it is easier for students to understand what iterations, operations and parameters are when they write and test programs using a universal programming language like Python. A teacher can introduce useful concepts like object and attribute applying the existing rich repertoire of media and class room activities. Metaphors from different domains might help to understand the interactive entities on the construction board. For example, a metaphor of a changeable line segment is a string, which can be connected to different things. The two end points of the string might change, but the string remains the same entity. In some respect a line on paper, drawn with pencil and ruler, is a worse metaphor of a SketchUp line segment.

There are basically two approaches to design instructive tasks to inspire computational thinking in the context of 3D-modeling. (1) A Task might focus on some functional features of SketchUp and stimulate to explore the computational structure of geometries. Example: “Create a triangular prism (Fig. 6 first picture) using only the following tools: rectangle, push-pull, select and move.” To solve this task the student needs to find out how to use the move-tool to move an edge. She or he might discover that moving the edge implies changing adjacent edges and faces and that the aggregate-structure of the model has been modified tacitly (the number of entities has been reduced).

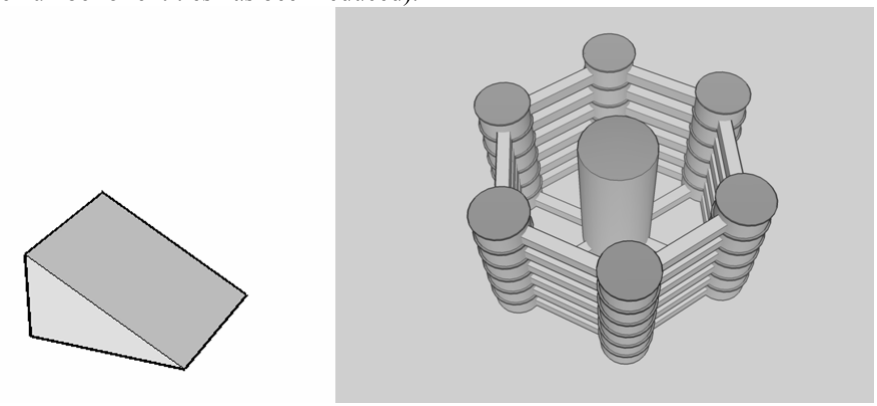

Fig. 6 Triangular prism (1) and a building with structured architecture (2)

(2) 3D-modeling projects imply situated learning [9]. A task may be – on the surface – an architectural or artistic design challenge. Still, it might *indirectly* encourage deploying and developing computational knowledge. Example: “Create a building like the one in Figure 6 (second picture)”. This can be done in minutes if you use components, the material- and the move-tool in a clever way. To construct it in a naïve way (line by line) is practically impossible.

And such experience might lead to an important insight: Informatics empowers you to create things.

References

1. Arzarello, F., Oliviero, F., Domingo, P., Robutti, O.: The Transition to Formal Proof in Geometry. In: Boero, P. (ed.) Theorems in School. From History, Epistemology and Cognition to Classroom., pp.305–323 (2007).
2. Ben-Ari, M., Yeshno, T.: Conceptual Models of Software Artefacts. Interacting with Computers 18(6), pp. 1336–1350 (2006).
3. diSessa, A. A.: Changing Minds. Computers, Learning, and Literacy. MIT Press, Cambridge, Massachusetts (2001).
4. Fischbein, E.: Intuition in Science and Mathematics. Reidel, Dordrecht Boston Lancaster Tokio (1987).
5. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns – Elements of Resuable Object-Oriented Software. Addison Wesley, Reading, MA (1995).
6. Hölzl, Reinhard: How does "Dragging" Affect the Learning of Geometry? International Journal of Computers for Mathematical Learning 1, pp.169- 187 (1996).
7. Jones, K.: Providing a Foundation for Deductive Reasoning: Students' Interpretations when Using Dynamic Geometry Software and Their Evolving Mathematical Explanations. Educational Studies in Mathematics 44, pp. 55–85 (2000).
8. Laborde, C., Kynigos, C., Hollebrands, K., Strässer, R.:Teaching and Learning Geometry with Technology. In: Gutièrez, A., Boero, P. (eds.): Handbook of Research on the Psychology of Mathematics Education: Past, Present and Future, pp. 275–304 (2006).
9. Lave, J., Wenger, E.: Situated Learning. Legitimate Peripheral Participation. University of Cambridge Press, Cambridge (1991).
10. Reenskaug, Trygve M. H.: Models – Views – Controllers. Technical Note. http://heim.ifi.uio.no/trygver/1979/mvc-2/1979-12-MVC.pdf (1979)
11. Schulte, C.: Duality Reconstruction – Teaching Digital Artifacts from a Socio-technical Perspective. Roland Mittermeir, Marciej M. Syslo (eds.): Informatics Education – Supporting Computational Thinking. ISSEP 2008 proceedings, pp. 110 – 121. Springer, Berlin Heidelberg New York (2008).
12. SketchUp Homepage: http://sketchup.google.com/. Accessed November 2008.
13. Weigend, M.: Intuitive Modelle der Informatik [Intuitive Models in Computer Science]. Universitätsverlag Potsdam, Potsdam (2007)
14. Wing, J. M.: Computational Thinking. Communications of the ACM Vol. 49, No. 3, pp. 33–35 (2006).