Blender for Architects 2013

Lukas Treyer, Gerhard Schmitt
Department of Architecture, Swiss Federal School of Technology, ETH Zurich, Switzerland

Abstract
In this paper I will discuss workflows of architects, the strengths and weaknesses of Blender in terms of these architectural workflows and ways to better support an architect in developing and/or visualizing a project. Among a general subjective observation of how architects use their tools and how the tools themselves are evolving at the moment, I will present a few propositions on the one hand how Blender could become more architect/design-friendly and on the other hand how a few interface concepts of Blender could enrich the development of architecture tools and what the first steps would be for that.

1 Introduction
This paper is aimed at Blender developers, Blender artists as well as architects and (computer) scientists involved in architectural and urban research and simulation. Therefore, it starts with a description of architectural workflows and Blender-specific terms are put in quotation marks. Among rather small closed feature proposition it contains also complex propositions that are interlinked with each other. This means its title “Blender for Architects 2013” refers also to possible design decisions to be taken in 2013 and may motivate developers to think about the potential of Blender beyond its fantastic animation capabilities.

2 Architects and the parametric
This section represents a subjective observation of my profession in order to give the non-architects among the readers an impression of architectural workflows. In order to do more scientific statements an in-depth field study about parametric tools would be necessary.
Throughout the last 30 years many efforts have been made to support and automate an architect’s workflow. But it’s been mainly in the last 10 years when the mouse started replacing the architect’s pen in architectural drawings. At that time “opinion leaders” in CAAD thought that “drawing at the computer”, where the computer simulates traditional design tools like a pencil, will be only an intermediate phase towards a new way of designing as we probably have it today with the popular Grasshopper-plugin for Rhino[2]. Still the drawing, the plans, remained the architect’s main communication tool, a language to communicate not only on the building site but also with clients and between architects themselves. With this 2D abstraction, an architect can focus on his/her conceptual thoughts or just on an area of a complex 3D design. By reducing the geometrical complexity, a plan can be packed with lots of context-specific information, e.g. for craftsmen or about energy consumption simulation etc. The scale actually doesn’t matter, plans and maps are always an abstraction, a strategy to reduce complexity in order to facilitate the design process and communication.
In general you will find as many different architectural workflows as there are architects. Even though it is possible to classify them to a certain extent, it is an architect’s job to react to the particular problems of each project and come up with strategies how to solve them. Sometimes it’s good to start with a 3D model, sometimes the qualities are defined by the material and one starts with collages instead. Sometimes the section of a building is very important and it is mainly developed in this “view" - as a drawing with a pencil tool. To support the important connection between drawings and 3D models, developers of the major architectural drawing suites started to provide architects with parametric tools, that should meet the needs of architects.
Dealing with complex problems, architects are trained to be precise but pragmatic. Ironically, it’s probably exactly due to this combination that at some point architects abandon the easy-one-click tools they are given by expensive architectural drawing suites, e.g. a wall tool or a window tool. Those tools get clumsy and very hard to control if you exceed a certain amount of precision because you cannot add new parameters but only adjust the existing ones. Many architects prefer raw polygons and lines over parametric tools.

3 Why Blender?
Blender implements a lot of ideas to support the strategy of “less predefined automation but more automation by interaction”[1]. Blender excels at giving users the tools they need to parameterize models on their own. In contrast to an application for architecture an animation software cannot only define a few parametric objects that are mainly used, like a window or a door, but it is designed
to let users create any shape they can imagine with almost any parameterization needed to facilitate animation. The tools for this in Blender are “Bone Armatures”, “Constraints”, “Modifiers”, “Shape Keys” and “Drivers(Scripts)”. There even have been some studies on ‘Shape Grammars’ using Blender’s “DupliFrames” capabilities together with its animation system[3]. Many of the ideas can be found in other 3D modelers and animation software, but it’s Blender’s openness that could help adapting those ideas to an architectural context. At the moment Blender is pretty hard for architects to learn on their own, because they have to learn a completely new interface concept, with only little reference to what they are used to (if they don’t know Maya). Yet, the user interface aims to be very transparent on what happens with the user’s geometry; no oversimplified black-boxes. By its nature – being developed as an animation software – Blender focuses on the creation of images and frames rather than 3D geometry. This might be a surprise to some of the readers but that’s pretty similar to the output of an architect’s work. It would be a keen understatement to reduce architecture to just 3D geometry. Most of the time an architect is drawing plans, a vector based image that relies on 3D data. Architects communicate with images, drawings and models, all of them having their own language, maybe even orthography depending on whom you talk to. Architects/designers “read” plans, they don’t just look at them.

In other words Blender’s 2D capabilities with its compositor and the recently added mask node, as well as the powerful UV-editor and of course its match moving capabilities, show that Blender addresses 2D equally to 3D. With additional features that would focus more on 2D printing it might get even more appealing to architects. With “Hook Modifiers”, a concept rather unique to Blender, 2D and 3D could be stitched together with every connection being adjustable by the user. Of course in a few decades we may not use printers anymore and buildings may be built by robots, and instead of printing 2D plans we will print 3D buildings. But even in such an environment vector graphics will not lose a single bit of their beauty, and hence expressive power when talking to clients and/or colleagues. Blender’s ability to link objects from different files into one scene not only allows its users to split the work but it does this very reliably. Consequently a shared workflow has been tested throughout many Blender open movie projects so far. Together with improved “hooks” this could be introduced in architecture as well.

4 Interdependent Propositions

The following propositions fully show their potential only if introduced together since they depend on each other. Their dependencies are printed in cursive style.

4.1 2D drawing

2D vector drawing and/or printing capabilities need to be mentioned here, as they repeatedly appear in the preceding sections. Many attempts have been made in form of a python add-on to add more CAD capabilities to Blender. Undoubtedly they are useful for tasks such as intersecting two edges. But when we want to draw a plan we need a 2D view window, which is on one hand ‘nothing more’ than just an orthographic top view of the 3D scene. On the other hand a 2D view could include a paper size, printing/PDF export (see figure 1), line width and color/transparency, and a templating system that not only stores line and polygon attributes in classes probably similar to CSS, but also drawing patterns, which could be stitched to the main drawing with improved “Hooks”, as described in section 4.3. Furthermore such patterns obviously could be used as hatches/fill patterns for faces and dot/dash patterns for lines.

Figure 1 Visibility control for printing in the outliner.

In the 2D-view, toggling between “Object Mode” and “Edit Mode” could be done with a double click on the object. If there would be objects hooked to parts of the geometry (vertices, edges, faces) a double click on those would toggle in their “Edit Mode”. If in such a hooked object there would be other hooked objects this could be repeated until there would be no hooked objects in an object anymore.

In “Object Mode” a bounding box around selected objects would be drawn allowing for scaling and rotation an object by mouse – and probably with touch enabled devices/displays in the future. The origin could be set with a mouse click, which is one of the reasons why snapping options should be strongly improved as well. Furthermore, mentioned for completeness, 2D boolean operations and the ability to import and mask raster images and PDF’s should be included. Of course all these tools would call for new keyboard shortcuts too, which is another reason to have 2D view and 3D view separated.

4.2 Pencil Tools

A pencil drawing tool would be necessary in order to offer at least one tool architects are familiar with. It would include a switch to draw lines, polygons, rectangles, circles with different methods; i.e. polygons with and without Bezier handles, circles with center/radius or with tree points etc. A pencil tool could be introduced in the 2D view later perhaps it could be introduced also in 3D if it makes sense.

Figure 1 Mockup of ‘Pencil Tools’ in 2D View
4.3 Hooks / Vertex Links

The existing “Hook Modifier” should be extended by the following features in order for the other propositions in this section to work properly:
- Hooks should not only link a vertex to an empty / object, but they should also allow for links between vertices of different objects directly, without the workaround of having an “Empty” in between. (An “Empty” in Blender is an object type, which is mainly used to control other objects and which has no geometry data. Think of it as a ‘point’-object that is more than just a vertex.)
- An “Object Constraint” that hooks objects to vertices or vertex groups should be introduced or the exiting “Copy Transformations” constraint should be improved and maybe renamed to ‘Hook’.
- Hooks should work in both directions. Linked objects and/or vertices should update each other. To avoid infinite update loops, this would only work if only one of the linked objects is active/selected.
- In order to make this work, hooks could hook to other hooks. If two objects have a hook with the same third object, the assigned vertices would stick to each other in the same position. This introduces a technique similar to ‘duck typing’ in modeling (drawing). It would leverage the use of a pattern library that can be created and managed like normal models and drawings. Since there are no prototype tests yet, it’s hard to estimate the potential of such a feature. The goal would be to let architects create their own parametric elements as they are given with traditional i.e. wall and windows tools from architectural drawing suites.
- With the development of a ‘Level of Detail’ feature in Blender, as explained in section 5.2, the visibility of each hook could be controlled with a level that would be set globally. In drawings for instance certain hooks could be visible only at scale 1:50 while and not at scale 1:200.

In order to not overcrowd the user interface with too many modifiers, the hooks panel should probably be densified. Similar to the material panel, hooks should be listed at the top of the panel with all the relevant information being shown only when a hook is selected. In order to make linked vertices, edges and faces visually more distinguishable from normal geometry, blue circles are proposed to highlight those links in the view port.

Finally, the introduction of hooks as proposed in this section would probably allow for new workflows in architecture, since different parts of a building or an urban design could be developed by different people in different files at different levels of detail/abstraction, while at the same time the possibility of importing them ‘in position’, e.g. without any tedious brute force hacks, would remain.

4.4 Node-based procedural drawing/modelling

With the “Node Editor” getting developed and used more and more (particles, mask nodes), we could think about introducing the strength of the “Node Editor” to modeling too, which could be labeled as ‘Non-destructive Editing’ or ‘History-aware Editing’. A very comprehensive example which some architects know already is Grasshopper for Rhino. Another example is Esri City Engine’s visual editor.

In Blender one way to achieve this could be to turn every tool from the “Tool Shelf” into a “Modifier”, with the modifiers themselves being a node. In order to not restrict existing workflows such a behavior could be turned on and off with the ‘Caps Lock’ key. While ‘Caps Lock’
being active all modifications in the 3D (2D) view port would be documented in the “Node Editor”; nodes would be automatically inserted while the user is modeling in the 3D view or drawing in the 2D view. The modifier settings could still be displayed in the “Properties Window” but the data flow would be controlled with a node setup. Instead of the “Add Modifier” dropdown in the modifier panel there would be a button called ‘Show Node Editor’ (Figure 4).

With “Object Constraints” and object references becoming nodes too, the node editor would evolve towards a parametric editor for everything. Node references to objects or vertices could be created with ‘Ctrl-Leftclick’ in the “Node Editor” with the corresponding elements being selected in the 3D/2D view port. The distinction between “Material-, Texture-, and Compositing-View” of the “Node Editor” might be abandoned since an “Object Node” would connect everything. Alternatively a fourth node editor view for modeling could be introduced, maybe together with the new “Particle Nodes System”, which is being developed at the moment by Lukas Toenne.

The potential of connecting it even with the “Logic Editor” for game controls will not be elaborated in this paper, but in order to unify all of Blender’s node editors the need of game developers would need to be included. The following sections and figure 6 explain the node setup in more detail. Figure 6 shows two sample node threes where on the most left side the original is shown and on the right the generated geometry. In the first example an additive approach is being used with some hole cutting in the end. The subdivision not only works for faces but also for lines as shown in figure 5 on the most right side. This would allow for dashed /dotted lines. Besides the ‘Face’ mode the modifier would also have a ‘Mesh’ mode where the whole mesh would be searched for intersections with either the hooked or duplicated objects. Perhaps this feature is rather a placeholder to stress the intention that with the new “Subdivision Modifier” parts of a mesh can get cut out. Alternatively one might add a ‘Replace’ option to the hooks, that would actually cut out the parts of the mesh to which it an object is hooked to. Tests with a prototype would surely help to investigate the important details of a node based split logic.

### 4.5 Subdivision-Modifier-Node

The “Subdivision Surface Modifier” could be extended to not only subdivide a mesh with the existing two modes “Simple” and “Catmull” but also with a more interactive ‘split logic’. Together with the new ‘hooks and node system’ similar results to split grammars [4] might be possible.

The idea is to divide a face in one direction into a specified number of tiles. The division would be graphically represented in the view port by a line, divided into segments. As shown in figure 5 the subdivision line would also have a dimensioning chain (red) as well as a number right next to the subdivision line (xx) that indicates the type and number of subdivisions. The distribution of the “Empties” could be controlled with splines (blue) similar to those we already know from the “Graph Editor”. With such ‘control lines’ we could probably not only control the location, but also the rotation and scale of the generated “Empties” (different colors; not shown in figure 5).

Figure 5 Splits in one and two dimensions on faces and a line (most right).

On order to control the length of each segment, we could enter the length’s individually in the dimensioning chain. The subdivision not only works for faces but also for lines as shown in figure 5 on the most right side. This would allow for dashed /dotted lines. The subdivision not only works for faces but also for lines as shown in figure 5 on the most right side. This would allow for dashed /dotted lines. Besides the ‘Face’ mode the modifier would also have a ‘Mesh’ mode where the whole mesh would be searched for intersections with either the hooked or duplicated objects. Perhaps this feature is rather a placeholder to stress the intention that with the new “Subdivision Modifier” parts of a mesh can get cut out. Alternatively one might add a ‘Replace’ option to the hooks, that would actually cut out the parts of the mesh to which it an object is hooked to. Tests with a prototype would surely help to investigate the important details of a node based split logic.

Figure 6 Node Setups for the creation of a procedural model with a distribution and a subdivision approach.
4.6 Distribution-Node

With the ‘Distribution Node’ the highly valued “Array Modifier” would get extended with ‘Dimensioning Chains’. Similar to the “Subdivision Modifier” we could distribute an object with control lines and/or by entering a number in the dimensioning or simply dragging it. It even might be a useful addition to distribute not only one but several already existing objects/vertices. This would allow for repositioning objects with a ‘Control Line’ in “Object Mode” or only vertices/edges in “Edit Mode”.

Moreover this would allow for more flexible workflows, since a designer could model in a non-procedural way and adjust the position/rotation/scale of objects or vertices later with a parametric tool. Instead of only ‘Control Lines’ one might think also of ‘Vertex Color Control’ where users could control the distribution of objects by painting on a map for instance. The control might be even handed over to a python script that integrates other data into the distribution process.

4.7 Mask-Modifier-Node

A modifier that enables black and white vertex painting on its output in order to “apply” only portions of the modified mesh which in turn can be edited in “Edit Mode” and which is linked to the rest of the mesh with the improved “Hooks”. It is very likely that an architect needs to edit exceptions of the parametric design conducted with modifier-nodes. Actually I suspect architects to only even use parametric design tools, if they are able to override their parametric rules manually, because there is always an exception; especially if you deal with renovation.

5 Independent Propositions

5.1 Curve-Modifiers

At the moment there are no special curve modifiers. Only with an Add-on the ability of ‘Lofting’ was introduced to Blender. Since it’s running only on python it might be interesting to have this functionality also available as a modifier. Especially a “Trim”-Function would be very appreciated I think since the Catmull-Clark subdivision modifier usually runs into problems when used together with a Boolean modifier.

5.2 Level of Detail

At the moment there are just a few workarounds that allow for Level of Detail in Blender. One very simple solution would be of course to have the same object at different levels of detail on different layers and then work with “Render Layers”. Another possibility would be to control a subdivision modifier with the distance between object and camera. Still, if we have a low resolution version of a window that consists only of a plane and an image of the window and a highly detailed model of it, a mesh subdivision modifier doesn’t fit well.

If we want a script or a node setup that selects the proper objects according to the LoD also the layer solution is not ideal, since there can be more than one layer visible, which might lead to unexpected results.

“Level of Detail” could not only be used in 3D to reduce complexity the further away from the camera an object is, but also in 2D to show different patterns at different scales.

5.3 Manual Extrusion of (2D) Curves

It’s a pity curves can only be extruded with a parameter and not directly in the 3D view port. It would be a good starting point to use curves more in Blender. Since “BMesh” unfortunately doesn’t support wholes in a polygon yet, we might push the NURBS features a bit in the meantime.

5.4 UV-Vertices with Shapekey support

Allowing for “Shape Keys” in the “UV-Editor” would enable us to animate textures. There is a patch by Pawel Kompak called “UV Offset Modifier” which allows you to move in U- and V- direction. Dealing with textures that need to be animated on a surface, this would be a great help.

5.5 Better API

Operators/tools/nodes should be easily accessible from Python in order to use the info window to learn scripting and not to start writing a second library that enables a Python scripter to properly extrude a face for instance. Calling an operator from Python means the operator should take mesh or object data as a parameter to, not only looking for selected objects and throwing exceptions if its context doesn’t fit.

It should be possible to create new nodes with Python (Cython). This would enable the community to actively help develop procedural modeling and drawing in Blender. As an example again Grasshopper for Rhino needs to be mentioned, where a lot of architects, artists and academics work with and on new nodes very actively.

6 Potential

Using Blender for a node based procedural modeling approach would enable artists and architects to use Blender’s powerful low-level parameterization tools to create higher-level templates that still can be adjusted to specific needs. Blender’s learning curve would get more linear for architects, since beginners could use templates from other artists and could dig into the low-level details once they feel up to.

An improved workflow to hook vertices and objects with vertices is not only the basis for a better interplay between 3D and 2D and also between nodes. It may improve modeling and rigging workflows in the field of animation as well.

The ‘Control Lines’ should be explored more, since they potentially provide a powerful high-level control inter-
face, that could give access to not only distribution and subdivision but also to other parameters implemented in Python. The ideal case would be to use Blender as a tool that supports an interactive evolution process: Find and define qualities of our design while having the design tested in simulations in real time. Maybe this could be accomplished with the node system too: We could include more and more data and make the nodes smarter and smarter. Therefore it might be worth thinking as well about how to link the CUDA/cycles ray-tracer with other nodes in order to use the GPU also to do something else than “just” light simulation, but maybe noise analysis, or fluid dynamics which in turn could speed up physic simulation processes in animation. By providing researchers a charming development framework with Python, nodes and interactive 3D editing there might be a revenue.

7 Literature


8 Appendix I

Figure 1 Overview on how the propositions are interlinked. Slide from the presentation at Blender Conference 2012.