

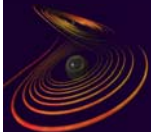
Managing Procedural Knowledge

Sven Havemann

Prof. Dieter Fellner

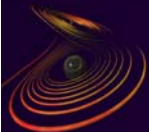
Institut für *ComputerGraphik*

TU Braunschweig, Germany



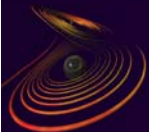
History: The success of the PC

- IBM PC released **1982** in response to Apple II
- PC market disregarded by IBM
 - ❖ *“individuals do not need a computer”*
 - ❖ *“a dozen mainframes are enough for the world”*
 - ❖ IBM ordered OS from Microsoft: DOS ...
- PC had overwhelming success
- Reason: **Killer application *Visicalc***
 - ❖ Executable of version 1 had ~29 KB
 - ❖ Text screen, no mouse – but useful and usable



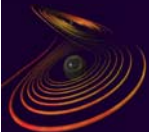
The Power of Spreadsheets

- Limited but well defined DOFs & OPs
 - ❖ Only **regular grid** of cells **A1, A2, ..., B1, B2, ...**
 - ❖ Expressions: **A3 = A1 + A2** (*dependency DAG*)
- Spreadsheets gave **power to average users**
 - ❖ Perform complicated computations **easily & instantly**
 - ❖ Make **non-trivial** use of a computer
- Calculate optimal prices for goods, try out variants, improve taxes and profit margins etc.
- Mixture: **Dynamic** values but **static** tables
 - ❖ Fundamental restriction
 - ❖ “declarative” rather than “procedural” (exceptions...)
- What if more flexibility is needed?



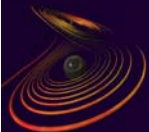
Resort: Programming

- Incentive: **Automization**
 - ❖ Even great tools must be used **interactively** (scale)
 - ❖ Deeply nested menus, lots of dialogue boxes
- Tools with built-in **Scripting Languages**
 - ❖ VBScript, EmacsLisp, MEL (Maya) ...
- Tools with **APIs** for C/C++ extensions
 - ❖ 3DStudioMax, Gimp, Mozilla, ActiveX, ...
- Problem: Few users have **programming** skills
 - ❖ Like learning a **foreign language**



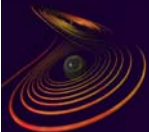
Procedural Knowledge (PK)

- **PK** is most valuable asset of
 - ❖ Individuals
 - ❖ Commercial companies
 - ❖ Academic institutions
- “*Definition*”: **PK** is the knowledge about
 - ❖ When and how to use a tool (e.g. software)
 - ❖ Order of processing steps to satisfy an order
 - ❖ Which tasks to assign to which individuals
 - ❖ What results to be expected for subsequent steps
- Very general, difficult to reason about **PK**



The Missing Link

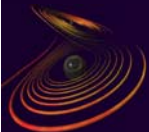
- PK representation: Programming Language
 - ❖ Generated manually or **automatically**
- Knowledge management: Tools that assist **Implicit** knowledge → **Explicit** knowledge
- Missing link: A technology for all users that
 - ❖ Permits to define a process
 - ❖ Allows **cross-application automization**
 - ❖ But does **not** require literal programming
 - ❖ Demands level of expertise like spreadsheets



The Code Generation Problem

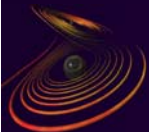
- Code generation in background requires a **simple** programming language!
- Consider C-code as **character string**
 - ❖ Complexity of software to generate C-Code
 - ❖ Complexity of tools to maintain C-Code

```
int function(int a, int b, (int)(*f)(int))
{
    int i,s=0;
    for(i=a; i<b; ++i) s += f(i);
    return s;
}
```



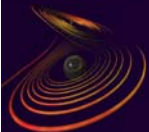
Simplistic Encoding of PK

- Operational view on PK as set of processes:
Process is a sequence of processing steps
- A processing step is an *operation*
 - ❖ **Constant operation:** Literal data are put on stack
 - ❖ **Elementary operation:** Takes data from stack, processes data, pushes results back
 - ❖ **Combined operation:** Pre-defined sequence, essentially a sub-routine
- Library of suitable sub-routines: **PK-DL**
 - ❖ Domain-dependent processes
 - ❖ Re-use of PK



Result: Stack-based Language

- Heavily under-rated as scripting languages
 - ❖ Tedious to program, stack acrobatics, ...
- But: Nobody wants to program!
 - ❖ Code generation in background is a nightmare with C/C++
- The **invisible language**: Adobe **PostScript**
 - ❖ **Unparalleled** in terms of quantity of generated code
- Closeness to pure data formats
 - ❖ Generalization of static data formats (sed/awk)
 - ❖ Exploit any sort of **regularity** in data, the effect of
1 2 3 4 5 6 7 is *identical* to **1 1 7 { } for**
 - ❖ More powerful by *orders of complexity*
- Theory: Kolmogorov complexity



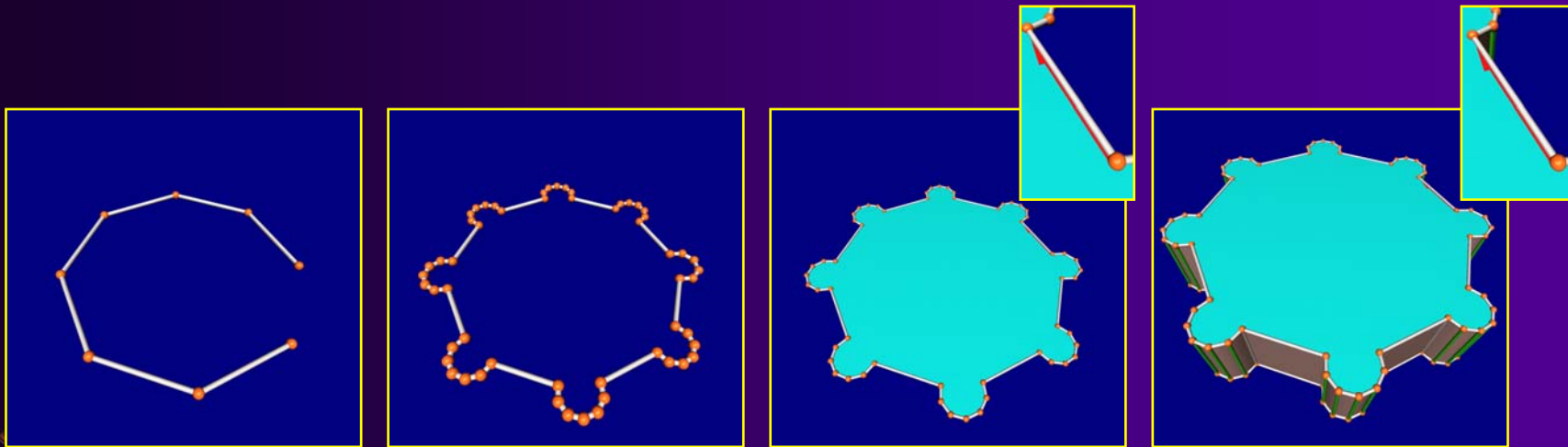
Generative Modeling Language

3D modeling works like an assembly line:

```
(0,0,0) (0,0,1) (2,0,0) 8  
0.3 (0,0,1) 8  
5  
(0,1,5)
```

```
circle  
corner-circles  
poly2doubleface  
extrude
```

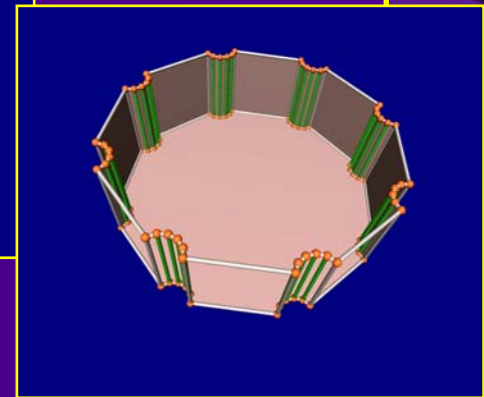
Separation of parameters & operations



From Objects to Operations

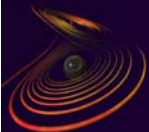
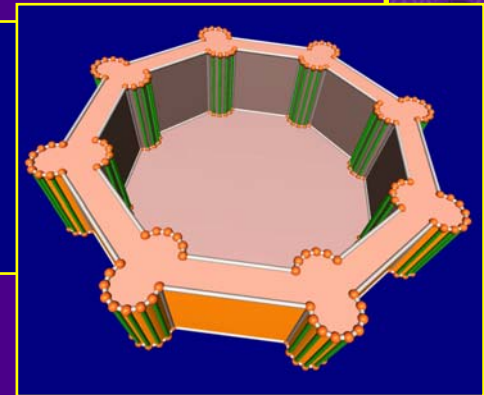
Next Step: Parametrization & Tools library

```
usereg !nrml !r !ex
(0,0,0) :nrml :r 8 circle
0.3 :nrml 8 corner-circles
5 poly2doubleface
:ex extrude
```

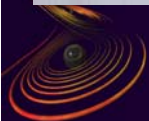


The key to re-using existing solutions:

```
(0,-1,5) (1.7,0,0) (0,0,-1) my-tool
(0,1,5) (2,0,0) (0,0,1) my-tool
killFmakeRH
```



Structural Similarity as PK



GML is stronger than XML

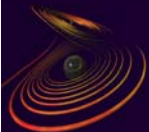
➤ Transition from XML to GML

```
<b> bold <i> italic text </i> bold again </b>  
b >bold< i >italic text< ni >bold again< nb  
b "bold" i "italic text" ni "bold again" nb
```

➤ These three examples are then equivalent:

- ❖ ` text `
- ❖ `pushfont +1 fontsize "red" fontcolor >text<
fontpop`
- ❖ `/myfont1 { pushfont +1 fontsize
"red" fontcolor } def
myfont1 >text< fontpop`

➤ **One** language versus XML+DTD+Java+...



Thank you
for your attention!

Check out for News on
www.generative-modeling.org

