Should We Integrate Programming Knowledge into the Architect's CAAD-Education?

Basic Considerations and Experiences from Kaiserslautern

Bernd Streich

University of Kaiserslautern
Planning Methods and CAAD in
Environmental Planning and Architecture
D - 6750 Kaiserslautern, Germany

1. Introduction

At the ECAADE-congress 1991 in Munich, the teaching concept of computer-aided architectural design of the faculty of architecture and environmental/urban planning at the University of Kaiserslautern has been presented. On that occasion, this brought about the question whether the curriculum should include programming knowledge. In this paper, the discussion shall be taken up again with several arguments in favour of the computer programming instruction. At first, a survey of the current discussion of the subject shall be given, then there will follow some reflections on the theoretical relationship between designing and programming, and finally, examples from the teaching experience in Kaiserslautern will be presented.

2. International Discussion

The following standpoints present several topical items of discussion:

- In the mid-1980s, E. Lee Kennedy wants architects to have programming knowledge or at least to be able to write macros [Kennedy 1986, p. 165 sq.].

- At a later date, William Mitchell, Robin Liggett and Thomas Kvan publish 'The Art of Computer Graphics Programming" mainly addressed to architects. In the introductory chapter, they justify to speak of an "art" with regard to computer programming by saying: "We treat computer programming for the production of graphics as an art form that (much like musical compositions, playwriting, and choreography) involves a distinction between specifying and performing a work. [...] The only way to become skilled in any art form is by doing. You must become familiar with the characteristics of your medium, and you must develop your skills through practice starting with simple exercises and gradually moving to more ambitious projects" [Mitchell et al. 1987].

- In the late 1980s, Mark L. Crosley summarized the problem that architects intensively use CAAD-systems but are absolutely unwilling to write programs themselves by the following question: "Who is the designer, the software author or

the software user?" Crosley advocates so-called "User Programming Languages" being an integral part of CAD-systems [Crosley 1988, p. 46 sq.].

- In 1990, "Logic Models of Designs" was published by Richard Coyne who, too, accentuates the major importance of programmming knowledge, and even supports the use of knowledge based systems that exceed the classical procedural programming methods: "Logic programming is a powerful tool for modelling design knowledge" [Coyne 1988, p. 61 and p. 29 sq.].

- At the ETH Zurich, the curriculum of the CAAD course taught by Gerhard Schmitt includes programming with AutoLisp in the AutoCad software environment. For Schmitt, too, the objective of teaching programming knowledge is "to prepare students for the difficult task of designing and changing CAD programs for the personal use or to the needs as specified by other architects" [Schmitt et al. 1991, p. 1; Schmitt 1988, p. 114 sq.].

As a summary, we can state that in the international discussion teaching programming knowledge for architects is considered to be a very useful addition to the architect's instruction. This opinion may meet with even more approval in future, because advanced programming knowledge will be needed to develop expert systems respectively knowledge based systems containing the architect's design repertoire and strategies to solve design problems.

3. Abstract relationships between designing and programming

Generally, designing can be understood as an innovative, creative and heuristic act of modelling respectively outlining objects or situations, transforming them into a new order unknown up to now. This is done in accordance with defined intentions, marginal conditions and assessment criteria. Two consequences emerging from the above definition can be exposed:

At first, designing concerns a process of finding one solution among an unlimited number of possible solutions. If you take a computer instead of traditional design instruments, this implies a certain repertoire of design elements permitting certain combinations. But the restriction to certain design methods leads also to a restricted number of possible solutions.

Therefore, CAD system engineers constantly try to ameliorate their software in order to make it more similar to the concept of designing. This might never be achieved perfectly, because no matter how perfect the system may be, some time the architect using the software will discover the limits of the system and is likely to exceed them in order to achieve something new and innovative. But he will not succeed unless he disposes of programming knowledge enabling him to realize innovative designing.

The second consequence is that the design process is essentially determined by design instruments. In architecture, traditional instruments are the crayon and several techniques of model building, which are in close connection to the sense of touch and other senses. These instruments influence the design process in the highest degree. Abstractly spoken, the classical design process represents a synchronous transfer of innovative creative associations of ideas into a visualized interpretation (= designing). Therefore, design models can differ very much in their artistic expressiveness.

The immediate combination between intellectual innovative activity and motoric behaviour which creates authenticity of design in this way - and perhaps that is the only

way to create authenticity (!) - is missing when the designing instrument is a computer. There is no motoric behaviour within the design process unless you think at the handling of the keyboard or of the mouse. The synchronous immediacy of mind and body, which has importance for giving design an individual expressive power, is absent.

Another important component of the design process is the communication between people in the background of the design process. Therefore, the use of the computer is limited. But we can only mention this aspect in this article without examining it more closely.

But where is the real design potential of the computer?

Among the characteristics of the computer, the nearly unlimited data storage capacity and its speed in executing programs undoubtedly are the most eminent. These characteristics can be used in order to visualize pictures or videos (e.g. the simulation of a motion process or photorealistic graphics) as it has been done, as well as to produce new shapes and structures which cannot be realized in the traditional way because of the tremendous volume of data (just think of fractals).

Designing, however, cannot be restricted to the formal production of new shapes and structures, since the architect's individual creative impulse is dominating. In addition to that, the architect does not only wish to use CAD systems but to extend and to modify them according to his own ideas. After all, this can only be done by programming.

Computer programs themselves are the result of planning and designing. The similarities between the planning aspect of writing computer programs and of other domains, especially of architecture and of urban planning, cannot be denied. In the 1960s, D. E. Knuth attached great importance to the idea that programming is a real art form ("The Art of Computer Programming") and in the 1980s, William Mitchell and his fellow authors shared this opinion, as it has been mentioned above. In this regard we find again analogies to the creative impulse of architects.

However, we can consider the relationship between designing and programming more closely under two aspects suggested by the following questions:

- Can designing be done by computer programming, i.e. is there a kind of programmed designing?

- Does only programming realize designing when computers are used?

At first, the conception of computers performing the design process as a whole and of computer programs executing design jobs seemed to be feasible, as Negroponte's "Architecture Machine" showed. Mainly in the 1960s and 1970s, several attempts were made to solve design problems by consequently using computer programs. By means of the information aesthetics of Birkhoff and Bense, even aesthetic problems seemed to reduce to a simple quantitative problem which is objectifiable.

Some of the earlier publications, however, are still worth reading today and recently, a few ideas were taken up again, slightly modified and under new terms. Unlike before, however, the expectations concerning the automation of the design process are less ambitious today, as the term "Design Support Systems", introduced some time ago, shows. Gerhard Schmitt considers "Design Support Systems" as a new generation of CAD-systems, revealing a basic tendency of the topical development of computer systems at all. (Among others, probably Terry Winograd and Fernando Flores' book

set off a major change in the conception of computer systems by claiming that computers should support communication between people. This thought, however, is not new, since already in 1961 Carl Adam Petri, a German mathematician, published his dissertation "Kommunikation mit Automaten", whose intentionally ambiguous title indicates a similar conception.)

Can designing be done by computer programs? Is programmed designing feasible and reasonable? These questions provoke scepticism. If only stereotyped shapes are to be produced or pure construction tasks are to be performed, involving geometrical, topological and constructive conditions, and if there are no more ambitious intentions, computer programs can be very helpful. But these computer programs do not perform the design on their own, and even very flexible CAD-programs support the architect's model only within the limits of the software conception. Designing by means of programs is restricted to a small domain - with regard to the above-mentioned definition. In future, creativeness and innovation will continue to be a human being's domain.

But this judgement must be modified, however, if you agree 1) on the point that programming itself is a designing act and 2) on the point that the designer of architecture actively uses the typical attribute of computers, i.e. that they are program based systems for the execution of his design ideas. Let us concentrate on CAD-systems.

As long as the CAD user is acting within the limits of the given CAD-system, the designer's concept is determined by the system. This may be tolerable as long as standard tasks and every day work are concerned. CAD users know, however, that different CAD-systems have different strong points and are not all equally suited for different purposes. CAD users know, too, that quite often small interface programs for data exchanges are needed. Frequently, users must write programs themselves because charging a software firm with the solution to the problem would be too expensive and would take too much time. But this kind of programming is a matter of technical routine and has nothing to do with design tasks.

Some time, architects wish to make use of the experience gained in the above described programming in order to facilitate the design process. This might be caused by a certain design idea that cannot be realized by the given programs or only by exceeding time and manual efforts.

Such problems can only be solved in a satisfactory way, if the CAD user can extend the given system according to his ideas and as he likes it. As a result, the term 'designing with computers' would better be replaced by 'designing by programming'. By the way, this would be logical and would correspond to the inherent principle of this instrument.

As an essential conclusion, we state that computer aided designing ends up in programming because only in this way new and original design concepts can be transferred from the designer's mind into the instrument computer. Therefore, given software systems must allow any extension you like.

4. Consequences for the CAAD-education of architects

Which consequences can be drawn from the above conclusion regarding the necessary programming knowledge and the programming environments at disposal? This question can be answered in general for all CAD users in architecture, but here it will be dealt with respect to the CAAD-education of architects.

Two basic requirements must he met in order to realize extensions to CAD-systems by designing programs:

- There must be an effective programming environment around the CAD-system allowing a simple programming of the design problem respectively allowing an adaptation or extension of the given system.
- The designer must be able to program himself design problems.

Regarding the first point, we wish to point out that architects are not to do the system engineer's job and develop a completely new CAD-system using C, PASCAL or FORTRAN. This may only be interesting for computer fans among the architects. But anyway, it is doubtful whether profit might outstrip expenditure or even whether the CAD-software market might accept new systems, economically spoken.

Accordingly, the programming environments being of interest are integral part of CAD-systems and allow any extension. Such CAD-systems which can be extended by software environments are characterized by

- being fundamentally open to extensions made by users;
- the possibility to integrate any CAD-command into the new program;
- the possibility to extent the CAD command repertoire by calling the new program.

Regarding efficiency and flexibility of such programming environments, you can differentiate between computer languages which can be subdivided into so-called macro-languages and into original user-oriented programming languages called 'User Programming Languages'.

Whereas macro languages only intend to write recurrent command sequences into a file - the macro being executed by calling the file - User Programming Languages are complete software environments. CAD-procedures written as macros take much time for running because they are often based on interpreter-languages. In contrast to macros, User Programming Languages are very similar to high-level programming languages such as PASCAL or C, and higher execution speed is obtained by executing a single compiling run when the complete program is translated into execution code before execution (in some cases, there is an additional link job linking the program with given modules of a library).

Some CAD-systems permit macro programming as well as programming within an original software environment based on a high-level language. Such systems provide high flexibility in using CAD-systems.

Concerning the second presupposition, we suggest for CAAD-education of architects in detail:

Programming knowledge should not be taught at the beginning of the CAAD-education because architectural students must develop basic skills in computer handling and in using application software first. Therefore, the University of Kaiserslautern offers programming courses only in the second degree (of altogether three degrees) [cf. Streich, ECAADE International Conference 1991 in Munich].

The teaching concept proved to be successful: including on the one hand fundamental methodical knowledge of structured programming and on the other hand small-step programming exercises whose results can easily be checked in the CAD-environment. The final result of the exercises, however, is the programming of a larger programming unit, extending the design applications.

One remarkable result is that students having obtained programming knowledge now assume a more critical attitude towards complementary application programs than before. The reason for this change is that now students are in a position to evaluate the efficiency and limits of the CAAD-system especially with regard to the application program's capability to realize their design conceptions.

5. Examples

In the following we show some programming examples carried out as computer-aided designing exercises.

DCAL-programming ("Data CAD Applications Language") - figure 1 and 2

- Complementary application program: CAAD-system SPIRIT (in Germany) respectively DataCAD (USA)
- Language characteristics: similar to PASCAL; compiling is possible
- Exemplary programming exercises: production of a Moebius-strip as an example of designing particular geometrical shapes and objects; integration into a building design model under SPIRIT; design manipulation of parameters (circle function into elliptic function etc.)
- Experiences: good acceptance by architectural students.

UPL-programming ("User Programming Language")

- Complementary application program: CAD system microCADDS/GCD
- Language characteristics: similar to PASCAL; compiling is possible
- Exemplary programming exercise: extension capability of mapping overlays based on two-dimensional ground plan data and storage of basic statistical information in files (extension from CAD to GIS)
- Experiences: high acceptance possible on condition that the CAD system provides more architectural applications.

SML-programming ("Simple Macro Language") - figure 3

- Complementary application program: Geographical Information System ARC/INFO
- Language characteristics: similar to DOS-batch; interpreter language
- Exemplary programming exercise: production of urban planning indices - "floor space index" and "site occupancy index" based on GIS-data
- Experiences: as soon as the necessity of macro programming became clear, students were eager to familiarize with the program; often they wished to get more programming knowledge, e.g. in C.

PASCAL-programming

- Complementary application program: no specific program

- Language characteristics: standard-PASCAL including object-orientation
- Exemplary programming exercise: experimental 'manipulation' of recursive functions for fractals
- Experiences: good accessibility even for beginners.

C-programming

- Complementary application program: no specific program
- Language characteristics: standard-C
- Exemplary programming exercise: programming of interfaces for data exchange (CAD data to DXF; GIS to CAD; cartographic data to GIS or CAAD)
- Experiences: only suitable for advanced learners.

SMALLTALK

- Complementary application program: none
- Language characteristics: standard-SMALLTALK
- Exemplary programming exercise: development of an architecture related expert system (application of case based reasoning on standard architecture)
- Experiences: only in cooperation with computer science department; architects participate in knowledge acquisition and system structuring.

6. Conclusions

After these considerations we can answer the question whether computer programming is the right domain for architects.

Taking into account that the architect's most important task consists of creating an environment in which human beings feel at ease and with regard to the fact that the architect tries to solve his design problem with the help of his mind, by sensitivity and by human communication, then there are many arguments against architect's programming and in favour of others being more competent in this domain.

However, leaving specific problems to others is dangerous because in future more expert knowledge will be part of computer-aided design systems in architecture.

Architects should participate in influencing this development and therefore they need certain knowledge, because you can only change what you understand. Should architects back out of this task and at the same time continue to use CAD-systems, then other ways of thinking would influence the specific way of thinking of architects. As it has just been stated in Neil Postman's most recent book, any technology involves the ideological tendency to influence thinking in a certain direction, and therefore scepticism and a good deal of criticism concerning technology are justified. If architects are not willing to work without computers, then they must not back out of influencing themselves this technology. Otherwise they would be alienated twice from the traditional design task: by the technology itself as well as by the software authors unfamiliar with architecture putting their stamp on this technology.

Which alternatives could be developed from this? By means of three theses, we wish to present the different standpoints and the pros and cons concerning programming knowledge for architects:

'Affirmative' thesis 1: CAAD-systems do not only help the architect to save time which he can use for creative designing or other innovative activities but also support creative designing itself; the architect starts creating software himself and profits from this knowledge for his design tasks.

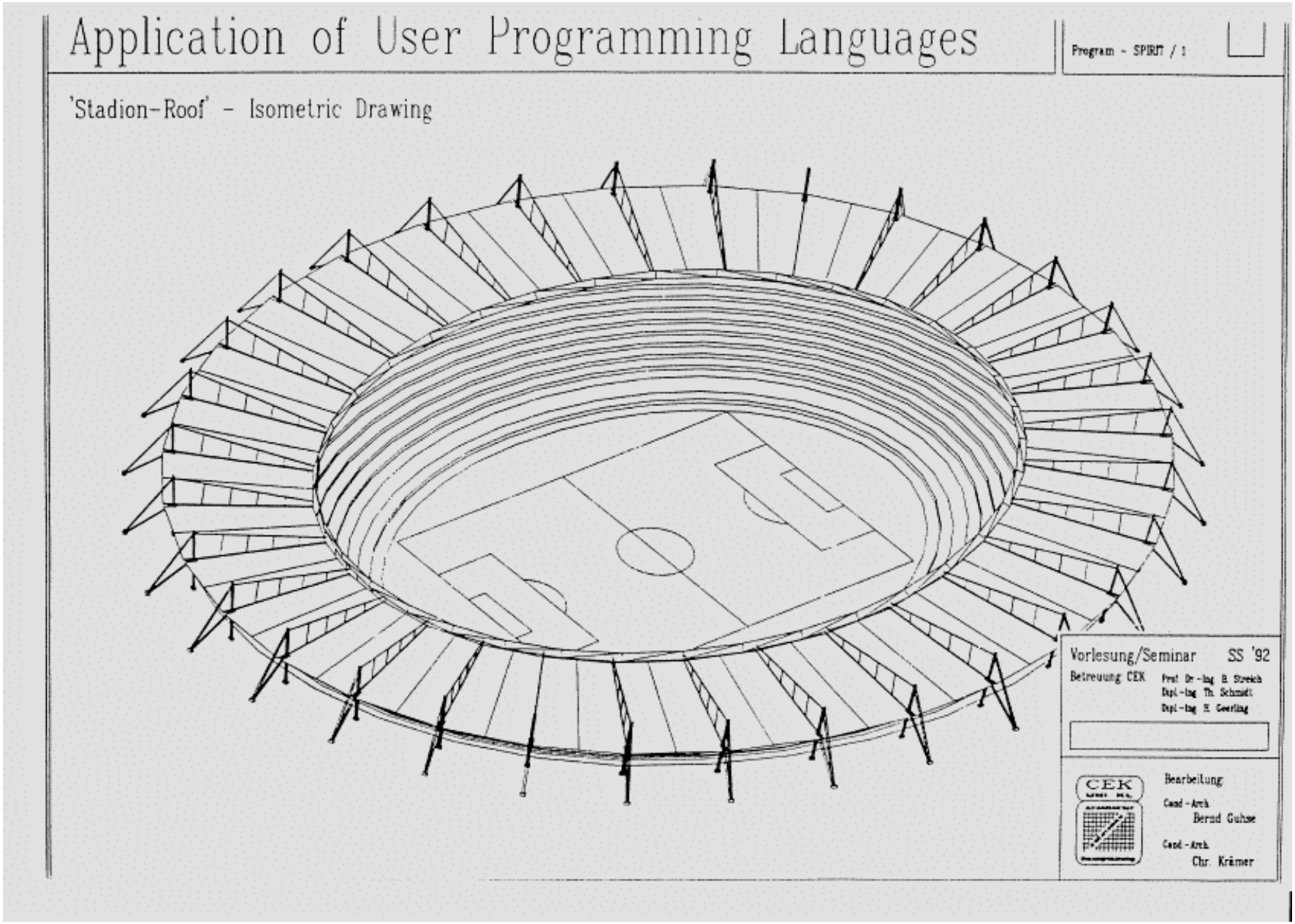
'Sceptical' thesis 2: It is true that architects use CAAD-systems, but the software authors are of other disciplines and the specific architectural aspect of designing will be dominated by other aspects; the architect's profession as such would be in danger if anybody were in a position to 'design' until authorization by means of CAD-systems based on integrated expert systems.

'Bifilar' thesis 3: On the one hand, architects make use of software systems mainly for day-to-day tasks as well as for innovative design which cannot be realized without programming, but on the other side they go on cultivating their individual authentical artistic by creating traditional design models.

Many sound arguments support thesis 3. We should offer our students every option which might be relevant for their future profession. Accordingly, the offer to obtain programming knowledge should not be omitted.

References

1. Coyne, R.: Logic Models of Design. London 1988.
2. Crosley, M. L.: The Architect's Guide to Computer-Aided Design. New York etc. 1988.
3. Kennedy, E. L.: CAD - Drawing, Design, Data Management. London 1986.
4. Knuth, D. E: The Art of Computer Programming. Reading/Mass 1960th.
5. Mitchell, W. et al.: The Art of Computer Graphics Programming. A Structured Introduction for Architects and Designers. New York 1987.
6. Postman, N.: Das Technopol. Die Macht der Technologien und die Entmündigung der Gesellschaft. Frankfurt a.M. 1992 (US original: "Technopoly", New York 1991).
7. Schmitt, G. et al.: CAAD Programmentwicklung 10-729 (Skriptum). Wintersemester 1991/92. Zürich 1991.
8. Schmitt, G.: Microcomputer Aided Design For Architects and Designers. New York 1988.
9. Winograd, T. und F. Flores: Erkenntnis - Maschinen - Verstehen. Zur Neugestaltung von Computersystemen. Berlin 1989 (US original "Understanding Computers and Cognition", Palo Alto 1986).


Application of User Programming Languages
Program - SPIRIT / 1
'Stadion-Roof' - Isometric Drawing
Vorlesung/Seminar SS '92
Betreuung CEK
Prof Dr-Ing B. Streich
Dipl-Ing Th. Schmidt
Dipl-Ing H. Geerling
CEK
Bearbeitung
Cand-Arch
Bernd Guhse
Cand-Arch
Chr. Krämer

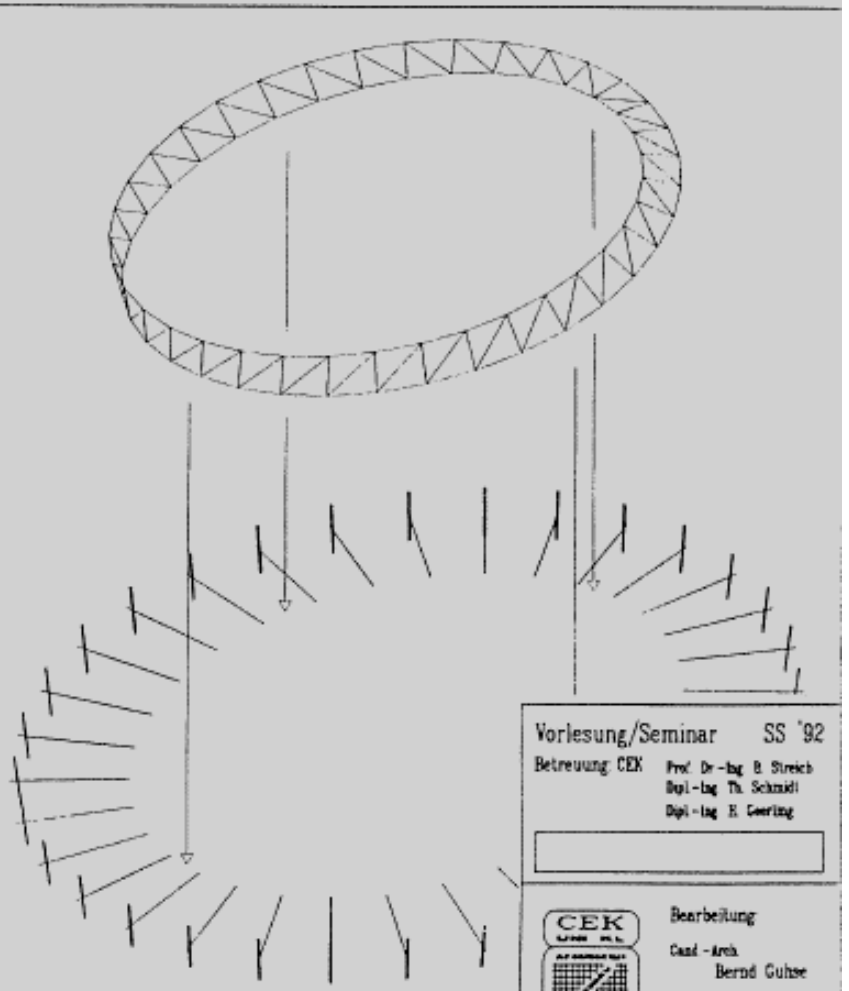
Application of User Programming Languages

Program - SPIRIT / 2

Program-Code (Extract!!)

```
!---HEADER-------------------------------------------------------------!
!   DCAL-Programm : 'Moebius'                                          !
!   Zweck : Programm zur Erzeugung eines ellipsenfoermigen Moebius-    !
!           bandes (mit Polygonen - Hidden-Line-Berechnung moeglich)   !
!   Vers. : Nummer 0.2                                                 !
!   Datum : Kaiserslautern, 11.08.1992                                 !
!---REALISATION cand.-arch. B. GUHSE / cand.-arch. C. KRAEMER----------!

PROGRAM Ellipse1;
   ....
   ....
   ....
   !------Unterprogramm 'Polygon'-----------------------------------------!
   PROCEDURE Polygon (p1,p2,p3,p4 : point);
           ....
           ....
           ....
           Ent_Init (ent,Entply);              ! Initialisieren Flaeche 1

             ent.PlyNpnt :=3;
             ent.PlyPnt  :=P1_Arr;
             ent.PlyIsln :=B_Arr;

           Ent_Add (ent);
           ent.color :=clrltgrn;
           Ent_Update (ent);
           Ent_Draw (ent,drmode_white);
           ....
           ....
           ....
        END Polygon;

   !------Hauptprogramm---------------------------------------------------!
   BEGIN

      WrtLvl ('Ellipse1');
      WrtMsg ('x-Koordinate fuer Ellipsenmittelpunkt eingeben : ');
      GetRll (x0);
      WrtMsg ('y-Koordinate fuer Ellipsenmittelpunkt eingeben : ');
      GetRll (y0);
      WrtMsg ('z-Koordinate fuer Ellipsenmittelpunkt eingeben : ');
      GetRll (z0);
      WrtMsg ('Radius in x-Richtung eingeben: ');
      GetRll (E_Rx);
      WrtMsg ('Radius in y-Richtung eingeben: ');
      GetRll (E_Ry);
      WrtMsg ('Banddurchmesser eingeben: ');
      GetRll (D);
      WrtMsg ('Winkelinkrement eingeben: ');
      GetRll (WG_I);
      ....
      ....
      ....
   END Ellipse1.   !-----------------------------------------------------!
```

Principle of Construction

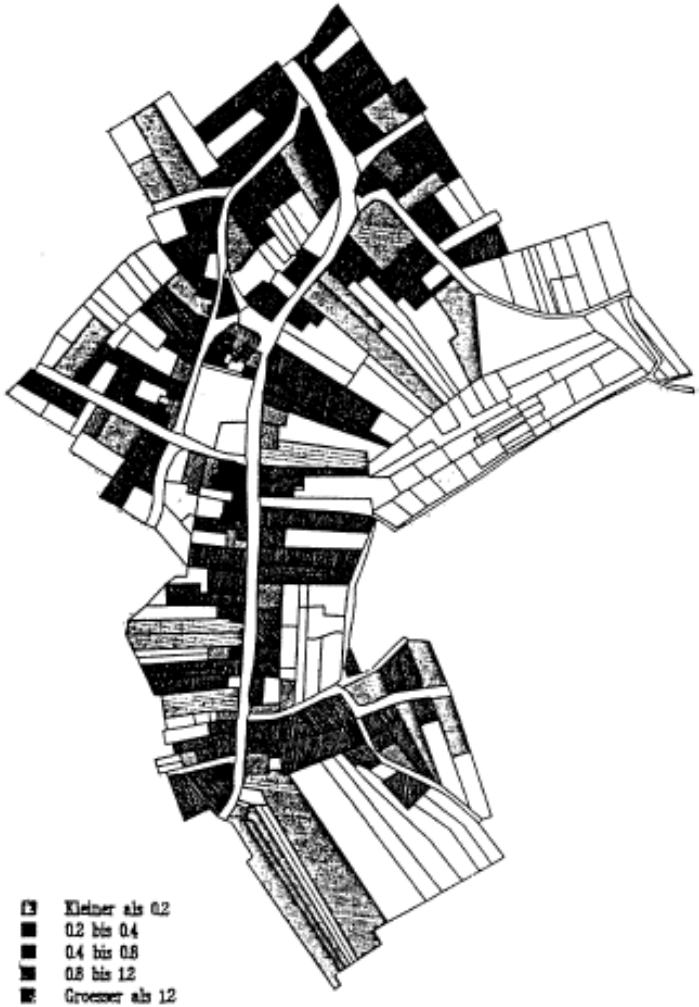


Ortsentwicklungsplanung Esthal

Bestandsanalyse

Plan Nr. 9 - ARC/INFO

Geschossflächenzahlen



- Kleiner als 0.2
- 0.2 bis 0.4
- 0.4 bis 0.8
- 0.8 bis 1.2
- Groesser als 1.2
- Sonstige Grundstuecke

Grundflächenzahlen

- Kleiner als 0.1
- 0.1 bis 0.2
- 0.2 bis 0.4
- 0.4 bis 0.8
- Groesser als 0.8
- Sonstige Grundstuecke

Programmcode (Auszug !!!)

Grosser Entwurf WS/SS 90/91

Betreuung CEK: Prof. Dr.-Ing. B. Streich
Dipl.-Ing. Th. Schmidt
Dipl.-Ing. H. Geerling

LOP: Prof. Dr.-Ing. H. Deanhardt
Dipl.-Ing. K. Ziegler

CEK

Bearbeitung:

Cand.-Arch. Bernd Guhse

Cand.-Arch. Rainer Kaiser