Self-Paced, Instructor-Assisted Approach to Teaching Solid Modeling and CAD

Pavel Solin[a,b]

[a] *University of Nevada, Reno, 1664 N Virginia St, Reno, NV 89557, USA*
[b] *NCLab, 450 Sinclair St, Reno, NV 89501, USA*

Abstract

We introduce an innovative approach to teaching solid modeling and CAD, where students actively engage in hands-on learning at their own pace, rather than passively listening to lectures. Throughout the course, their 2D and 3D models are evaluated in real-time by an advanced cloud computing platform that provides instant feedback, helping them build confidence and master the subject more effectively. Instead of lecturing, instructors dedicate their time to providing individualized support. We also explain why scripting-based CAD simplifies the learning process compared to traditional CAD software and why we have chosen the Python-based Programming Language of Solid Modeling (PLaSM). A key focus is the automated server-side grading of student assignments, a cornerstone of the self-paced course, offering a level of structured guidance and feedback that traditional CAD software lacks.

Keywords: Computer-Aided Design (CAD), Solid modeling, Constructive solid geometry (CSG), Career technical education (CTE), Competency-based education (CBE), Self-paced learning, Asynchronous learning, Learning by doing

1. Introduction

Over the past decade, we have taught solid modeling and CAD to a diverse range of learners, from high school CTE students to adults looking to upskill in the age of automation and Industry 4.0. Initially, we used traditional CAD software such as Blender and SketchUp. While these tools are

Email address: solin@unr.edu, pavel@nclab.com (Pavel Solin)

simpler than commercial software like SolidWorks or AutoCAD, many students still struggled with them. The challenge was not just in understanding 3D modeling concepts, but also in navigating complex graphical user interfaces (GUIs). Students often found themselves focusing more on learning the software's interface rather than the fundamentals of 3D modeling itself.

To address this challenge, we transitioned to a scripting-based CAD approach. After evaluating several options, we selected PLaSM (Programming Language of Solid Modeling), which we later customized based on student feedback, as detailed in Section 7. We found that scripting significantly enhanced the learning experience for most students. In fact, it is hard to imagine a lower barrier to entry than simply typing two lines of code, as demonstrated in Fig. 1.

```
c = CUBE(2)
SHOW(c)
```

Figure 1: Minimal PLaSM script to create and display a $2 \times 2 \times 2$ cube

The corresponding 3D model is shown in Fig. 2.

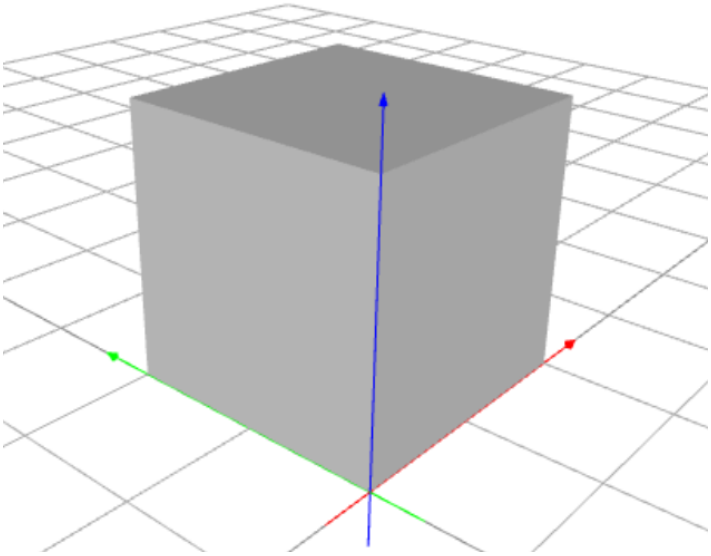

Figure 2: The corresponding 3D model, computed on the server and displayed back in the student's web browser

This simplicity eliminated both perceived and real barriers to entry, allowing

many students to truly enjoy creating 3D models for the first time. Additionally, scripting-based CAD proved to be more effective in teaching geometric transformations and Boolean operations fundamental to Constructive Solid Geometry (CSG). Our observations align with findings from other studies (see, e.g. [1, 2, 3] and the references therein), which suggest that integrating scripting into CAD education not only enhances students' learning experiences but also fosters critical thinking and problem-solving skills, providing a more comprehensive approach compared to traditional CAD software.

However, another significant challenge remained: the limitations of traditional synchronous classroom instruction. In this paper, we examine the difficulties our students faced and how these challenges were addressed through the development of a self-paced solid modeling course. The key innovation of this course is its real-time server-side grading of student assignments, which we explore in detail. We also provide an in-depth explanation of the course design, with the full syllabus available in Appendix A.

The outline of the paper is as follows: Section 2 provides background information on solid modeling and Constructive Solid Geometry (CSG). Section 3 discusses common challenges in teaching solid modeling and CSG. Section 4 highlights the benefits of self-paced, instructor-assisted learning. Section 5 presents an overview of scripting-based CAD tools. Section 6 introduces the Programming Language of Solid Modeling (PLaSM). Section 7 details how we customized PLaSM for educational purposes. Section 8 explains the implementation of automated server-side grading, with a concrete example in Section 9. Section 10 summarizes our conclusions. Finally, Section Appendix A presents a sample syllabus for an introductory 3D modeling course used with our students.

2. Solid Modeling and CSG

Solid modeling is a fundamental technique in Computer-Aided Design (CAD), engineering, and computer graphics that provides a mathematically precise and volumetric representation of 3D objects. Unlike surface modeling, which only defines an object's outer shell, solid modeling ensures that objects are watertight, physically accurate, and suitable for manufacturing, simulation, and analysis [4, 5]. Solid modeling is widely utilized across industries such as advanced manufacturing, construction, architecture, mechanical design, 3D printing, and computational simulations, where it enables engineers to create, modify, and analyze complex structures efficiently. It encompasses

several techniques, including boundary representation (B-rep), sweep-based modeling, and feature-based parametric design [6, 7].

A key subset of solid modeling is Constructive Solid Geometry (CSG), which constructs complex 3D shapes using Boolean set operations such as union, intersection, and difference applied to simple geometric primitives like cubes, spheres, and cylinders. These primitives are typically represented through their boundaries, including implicit surfaces, and combined using hierarchical structures known as CSG trees [8, 9]. CSG has gained prominence in scientific and engineering applications due to its compact representation and computational efficiency. Unlike boundary representation (B-rep) methods, which store explicit surface descriptions, CSG facilitates robust Boolean operations that maintain topologically consistent solid models and eliminate ambiguities often associated with boundary-based approaches [10, 11].

One of the main advantages of CSG is its parametric and procedural nature, which allows for easy modifications and adaptability in design processes. This characteristic is particularly valuable in CAD systems where iterative design changes need to be implemented efficiently [12, 13]. Additionally, CSG supports exact geometric queries, making it suitable for engineering simulations, finite element analysis, and computational physics applications [14, 15]. Many CAD platforms incorporate CSG principles to facilitate mechanical design, architectural modeling, and product engineering due to its efficiency in constructing complex structures from simple primitives [16, 17].

CSG also plays a significant role in computer graphics and visualization, where it is used in real-time rendering, ray tracing, and volume rendering to create complex objects with well-defined intersections and unions of shapes [18, 19]. Moreover, additive manufacturing processes benefit from CSG models by ensuring constructability and enabling the efficient generation of support structures [20]. Many computer game engines utilize CSG techniques for level design, collision detection, and physics-based simulations [21, 22]. Furthermore, CSG-based representations are widely used in scientific computing to define domain geometries for simulations involving fluid dynamics, heat transfer, and structural analysis [23, 24].

3. Challenges Associated with Teaching Solid Modeling

It is well known that teaching solid modeling and Constructive Solid Geometry (CSG) presents a significant challenge in Career Technical Education (CTE), engineering, and computer science curricula [25, 26, 27]. The

complexity of the subject stems from its reliance on spatial reasoning, mathematical formulations, and algorithmic implementations [28]. While solid modeling is fundamental to Computer-Aided Design (CAD), computational geometry, and manufacturing processes, students often struggle with abstract concepts, Boolean operations, and hierarchical modeling structures [29].

Various pedagogical approaches have been adopted to teach solid modeling effectively, including the use of interactive visualization tools and CAD software that allow students to experiment with Boolean operations and observe real-time results [30, 31]. Additionally, teaching the underlying mathematical principles, such as set theory and boundary representations, helps students develop a deeper conceptual understanding [32, 33]. Engaging students in hands-on modeling projects reinforces learning and enhances problem-solving skills [34, 35]. Recent advancements also incorporate VR-based solid modeling environments to improve student engagement and comprehension [36, 37].

Despite these approaches, teaching solid modeling and CSG remains challenging for several reasons:

1. First and foremost, students often find the Graphical User Interfaces (GUIs) of advanced CAD tools overwhelming, regardless of whether they are free or commercial [38, 39]. As a result, they frequently conflate learning solid modeling and CSG with learning how to navigate a specific CAD software product. Over the past decade, many CTE teachers have reported that the complexity of these software tools is one of the primary reasons why students develop a negative attitude toward the subject [40].
2. Most teaching approaches rely on synchronous learning, requiring all students to progress at the same pace. This often leads to faster students becoming idle and disengaged, while slower students struggle to keep up with both the instructor and their peers [41]. Studies have shown that self-paced learning methods can significantly improve student outcomes in CAD education by reducing cognitive overload and allowing for personalized learning experiences [42].
3. Another significant drawback of traditional classroom-based instruction in solid modeling and CSG is the lack of hands-on practice. Mastering this subject requires both conceptual understanding and practical skills; however, the amount of hands-on experience most students receive is often insufficient [28, 43]. Research suggests that active learn-

ing strategies, such as project-based assignments and scripting-based CAD, enhance skill acquisition and long-term retention in engineering education [44].

4. The abstract nature of Boolean operations, hierarchical tree structures, and the mathematical foundations of CSG further complicate the learning process [45, 46]. Studies have demonstrated that students benefit from explicit instruction on these topics using procedural modeling and algorithmic thinking [47, 48].
5. Finally, the significant variability in teaching methodologies across institutions leads to inconsistencies in learning outcomes [26, 29]. Standardizing CAD education with scripting-based approaches and automated assessment methods has been proposed as a solution to improve consistency and scalability [49].

In this paper, we present a novel method for teaching solid modeling and CSG that aims to address points 1–3 outlined above. We propose an approach that leverages a simple and intuitive scripting-based CAD system, eliminating the burden and frustration associated with learning the GUI of a complex CAD platform. Our method is self-paced, allowing students to progress at their own speed while their work is automatically evaluated and assessed in real-time by an advanced software platform [50].

Instead of passively listening to an instructor, students engage in carefully designed, bite-sized tutorials, examples, exercises, and graded tasks, learning one concept at a time. They must demonstrate mastery of each concept before advancing to the next stage. This approach fosters confidence, leads to superior learning outcomes, and most importantly, enables students to genuinely enjoy learning solid modeling and CSG.

4. Self-Paced, Instructor-Assisted Learning

We have already discussed the advantages of using a scripting-based CAD system over traditional CAD software for teaching solid modeling and Constructive Solid Geometry (CSG). However, another critical challenge remains: the limitations of traditional synchronous classroom instruction.

Traditional synchronous classroom instruction often fails to accommodate the diverse learning paces of students. In these settings, all students are expected to progress through the material simultaneously, which can disadvantage both those who grasp concepts quickly and those who require more

time. This rigid structure can lead to disengagement, as faster learners may become bored, while others may feel overwhelmed. Research indicates that self-paced learning allows individuals to progress at their own speed, catering to their unique needs and significantly improving overall learning outcomes [51, 52, 53].

Furthermore, the average student's attention span presents another challenge in traditional classrooms. Extended periods of passive instruction can reduce concentration and hinder information retention. Studies show that attention spans in learning environments typically decline after 10-15 minutes, necessitating instructional designs that encourage active engagement [54, 55]. Self-paced learning addresses this issue by enabling learners to take breaks as needed, revisit difficult concepts, and structure their study sessions in alignment with their cognitive capacities. This flexibility not only enhances engagement but also improves knowledge retention [56, 57].

In addition, self-paced learning fosters the development of self-regulation and time management skills. By taking ownership of their educational journey, students decide when and how to engage with the material. This autonomy encourages deeper comprehension and cultivates lifelong learning habits. Studies have shown that self-paced learners often exhibit higher retention rates and develop effective learning strategies that extend beyond the classroom [58, 59, 60].

For these reasons, we initially introduced an alternative instructional model in the context of Linear Algebra [61], where students engage in active learning 100% of the time, supported by an advanced interactive learning platform and real-time instructor assistance. The overwhelmingly positive student feedback validated this approach. Its success prompted us to extend the methodology to teaching Python programming [62] and SQL [63], where it proved equally effective. To avoid redundancy with our previous publications, we provide only a brief summary of the teaching method below.

Instead of passively listening to lectures or struggling to learn independently without guidance, students actively engage in learning at all times under the supervision of an instructor. Using their own devices, they progress at their own pace through bite-sized tutorials, examples, exercises, and graded practical tasks. Their work is continuously evaluated by a cloud-based platform that provides real-time feedback, instantly validating their understanding and offering assistance when needed. To advance, students must demonstrate mastery of each concept before proceeding to the next.

Instead of delivering lectures, the instructor provides individualized sup-

port, significantly improving student outcomes and offering greater insight into each learner's progress and areas of difficulty. After more than a decade of implementing this approach, we are convinced that nothing can replace the value of one-on-one interaction between students and their instructor.

5. Scripting-Based CAD Tools

Numerous educators, ourselves included, prefer using scripting-based CAD tools to teach solid modeling and Constructive Solid Geometry (CSG). Examples of such tools include PLaSM [1] and OpenSCAD [64]. These tools emphasize scripting (typing simple code or pseudo-code) over the use of a GUI. Students type simple commands to define basic 2D and 3D shapes and apply geometric transformations and Boolean operations to them while practicing elementary computer programming concepts.

It is well known that engaging in scripting and programming activities can significantly enhance attention to detail. The meticulous nature of writing code requires individuals to focus on syntax, logic, and structure, thereby fostering precision and thoroughness. Research indicates that the structured environment of programming encourages the development of problem-solving skills and attention to detail [65]. For instance, a study published in the *Journal of Educational Computing Research* found that students who participated in coding exercises demonstrated improved analytical thinking and precision in their work [66]. Additionally, the process of debugging code, i.e., identifying and correcting errors, further reinforces meticulousness, as it demands careful examination of each line of code to ensure accuracy and functionality [67, 68]. Therefore, incorporating scripting into educational curricula or personal development plans can be an effective strategy for enhancing attention to detail.

Another benefit of using a scripting CAD is that the written commands require concrete values for sizes, distances, angles, and vectors. In other words, students must calculate the correct values either mentally or on paper before entering them into the script. This contrasts with using a GUI, where they can often manipulate objects visually using a mouse without needing to compute precise numerical values [69].

However, improving attention to detail and quantitative geometry skills are not the only advantages of using a scripting CAD over a GUI. The script represents a detailed step-by-step history of all the shapes the student defined, along with all the transformations and Boolean operations they ap-

plied. This means that by reviewing the script, an instructor can easily identify mistakes or inefficiencies in the student's design thinking and workflow. In contrast, when a student uses a GUI, the instructor can only assess the final model, without insight into the thought process and the steps taken to arrive at the solution [70].

Another distinct benefit of scripting CAD tools is that they are conducive to implementing automated server-side grading of student designs. Receiving real-time feedback on their work is crucial for students, as it enables them to immediately identify and correct mistakes [71]. This capability is a cornerstone of our self-paced approach to teaching solid modeling and CSG, and we will discuss it in detail in Section 8.

Research has also explored how integrating programming with CAD enhances students' ability to comprehend geometric reasoning and algorithmic design, making such tools valuable in engineering and computer science curricula [72, 73, 35]. Additionally, studies have demonstrated that scripting-based CAD fosters computational thinking, allowing students to develop a deeper understanding of design principles and logical structuring in 3D modeling tasks [74, 75].

6. Programming Language of Solid Modeling (PLaSM)

A prominent position among scripting CAD tools is held by PLaSM (Programming Language of Solid Modeling) [1]. This is an open-source, Python-based scripting language developed by Alberto Paoluzzi and his research group at the University of Roma Tre in Italy. PLaSM is a functional programming language specifically designed for computational geometry and parametric solid modeling. It provides a powerful and expressive framework for creating complex geometric models using mathematical functions and Constructive Solid Geometry (CSG) operations.

PLaSM is based on a higher-order functional programming paradigm, utilizing a Lisp-like syntax with a strong emphasis on geometric abstraction and declarative modeling [76, 77]. It supports Boolean operations such as union, intersection, and difference and enables the definition of models through variables and functions, facilitating flexible design modifications. Unlike traditional CAD systems that rely on explicit modeling, PLaSM promotes parametric and algorithmic design, making it particularly effective for procedural geometry and generative design applications [78].

One of PLaSM's key advantages is its support for n-dimensional geometry, allowing complex transformations and hierarchical modeling [79]. This capability makes it well-suited for advanced geometric computing, topological modeling, and spatial data representation [80]. Additionally, PLaSM can interface with external tools for CAD applications and finite element analysis (FEA), making it a valuable tool in engineering simulations and structural analysis [81].

PLaSM is widely used in academic and research environments for teaching computational geometry, solid modeling, and parametric design [82]. Its applications span architectural design, where it is employed for generating parametric building models, urban planning simulations, and digital fabrication workflows [83, 84]. In engineering and manufacturing, PLaSM supports the modeling of mechanical components, structural frameworks, and 3D-printable objects, enhancing CNC machining workflows and rapid prototyping processes [85]. Moreover, PLaSM has been applied in scientific data visualization across multiple disciplines, including physics, biology, and computational sciences, to model complex spatial structures and dynamic simulations [86, 87].

Importantly, PLaSM serves as a valuable educational tool in colleges and universities, supporting courses on computational design, algebraic geometry, functional programming, and algorithmic modeling [88, 89]. As an open-source scripting CAD, it provides students with hands-on experience in parametric modeling, geometric reasoning, and computational design thinking, making it an excellent platform for engineering, architecture, and computer science education.

7. Modifying PLaSM for Educational Purposes

While working with students, we observed that the functional programming paradigm used in PLaSM introduced an unnecessary learning challenge for many of them. The higher-order functional approach, though powerful, required students to adopt a mathematical and abstract way of thinking that was often unfamiliar and difficult for beginners in computational design. Additionally, students found the lack of color representation frustrating, as all models in the original PLaSM were displayed in grayscale. This absence of color made it more difficult to distinguish geometric features and spatial relationships, which are essential for intuitive learning.

To address these issues, we implemented a procedural wrapper around the original PLaSM. This modification reduced the generality from supporting arbitrary dimensions to focusing solely on 2D and 3D modeling. It also introduced color support, simplified several commands, and made PLaSM overall more intuitive and user-friendly for students. By adopting a procedural rather than purely functional approach, students could define models using step-by-step commands, making the learning process more accessible while preserving the expressive power of the language.

For illustration, a sample 3D model consisting of a $6 \times 6 \times 6$ cube with a horizontal hole of radius 2 units at its center is shown in Fig. 3. The coordinate axes follow the standard color coding convention: XYZ = RGB, meaning X is represented in red, Y in green, and Z in blue.

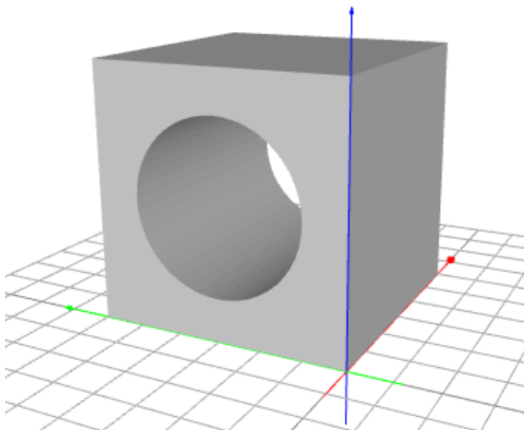

Figure 3: Sample 3D model

The corresponding PLaSM script, which adheres to the original functional programming paradigm, is shown in Fig. 4.

```
cube = CUBOID([6, 6, 6])
hole = CYLINDER([2, 6])(64)
hole = R([1, 3])(PI/2)(hole)
hole = T([1, 2, 3])([0, 3, 3])(hole)
hollow_cube = DIFFERENCE([cube, hole])
VIEW(hollow_cube)
```

Figure 4: Corresponding script written with the original PLaSM

Some aspects of the script require further explanation. To begin with, the original PLaSM operates in any dimension, which is why it uses the keyword `CUBOID` instead of dimension-specific keywords such as `SQUARE` or `CUBE`. Additionally, most functions in PLaSM are designed to accept a single parameter, necessitating extra brackets in `CUBOID([6, 6, 6])` rather than allowing a direct input like `CUBOID(6, 6, 6)`.

The functional programming paradigm in PLaSM relies heavily on functions that return other functions. For instance, on line 2, `CYLINDER([2, 6])` defines a cylinder with a radius of 2 units and a height of 6 units. This expression returns a function, which then receives an additional argument, `64`, specifying that the cylindrical surface should be approximated using 64 planar facets. In other words, the base of the cylinder is a regular polygon with 64 sides.

A particularly interesting case arises on line 3, where the `R` command represents rotation. In 3D, one would typically expect a single axis to define the rotation. However, in four or more dimensions, a rotation must be defined by specifying two axes that span a plane, with the rotation occurring perpendicular to this plane. Consequently, typing `R([1, 3])` results in a rotation about axis `2` (the Y-axis). This expression returns a function that takes an angle in radians, in this case, `PI/2`. That function, in turn, returns another function that finally receives the object `hole` and applies the rotation. We found that this hierarchical function application was often confusing for students, and the use of radians instead of degrees further complicated their understanding.

On line 4, the command `T([1, 2, 3])` represents a translation along the X, Y, and Z axes, in that order. This expression returns a function that takes another argument, `[0, 3, 3]`, which specifies that the object moves by 0 units in the X direction, 3 units in the Y direction, and 3 units in the Z direction. Again, this returns another function that ultimately receives the `hole` object and applies the translation.

It is important to note that both `R` and `T` return a new 3D object rather than modifying the original one in place. As a result, modifying an object requires overwriting the old variable with the transformed version. Many students found this unintuitive, as they expected transformations to be applied directly to the original object.

Finally, on line 5, the object `hole` is subtracted from the object `cube` to create the final shape. This operation generates a new object rather than modifying `cube` in place, which would have been the more intuitive behavior

for many students.

The same script, rewritten using our procedural wrapper, is shown in Fig. 5.

```
cube = CUBE(6)
hole = CYL(2, 6)
ROTATE(hole, 90, Y)
MOVE(hole, 0, 3, 3)
SUBTRACT(cube, hole)
SHOW(cube)
```

Figure 5: The same script written with our procedural PLaSM wrapper

Note a few things:

- On line 1, defining a $6 \times 6 \times 6$ cube is simpler and more intuitive by typing `CUBE(6)` instead of `CUBOID([6, 6, 6])`.
- On line 2, the value `64` is now a default parameter, reducing confusion for students. If a finer resolution is needed, it is still possible to specify it explicitly, for example, by typing `CYL(2, 6, 128)`.
- Object `hole` is rotated in place, meaning that no new object is created. Typing `ROTATE(hole, 90, Y)` is simpler and more intuitive than `R([2, 3])(PI/2)(hole)`. Additionally, all angles have been converted from radians to degrees to further simplify usage.
- On line 4, object `hole` is translated in place, ensuring that no new object is created. Typing `MOVE(hole, 0, 3, 3)` is simpler and more intuitive than `T([1, 2, 3])([0, 3, 3])(hole)`.
- The function `SUBTRACT` modifies the object `cube` in place, providing a more intuitive syntax compared to `DIFFERENCE([cube, hole])`.
- Syntax highlighting has been introduced to help students easily identify typos and improve code readability.

To summarize this example, while the functional programming paradigm may not present a challenge for an experienced computer scientist, it proved to be a significant hurdle for our students, many of whom had limited programming experience. We hope the reader now understands why we felt

compelled to develop a procedural wrapper for PLaSM and simplify its usage for students.

More details on the original PLaSM language can be found in [1]. The procedural wrapper is freely available on GitHub [90], and numerous examples illustrating its use can be found in the free textbook [91].

8. Automated Server-Side Grading of Student Assignments

Now that the reader has a clear understanding of how 2D and 3D models are created with PLaSM, we turn our attention to automated server-side grading of student assignments. As mentioned in Section 4, receiving real-time feedback is the cornerstone of the self-paced 3D modeling course. The basic idea behind this approach is straightforward.

Consider a scenario where a student's task is to create a 3D model named `design`. While developing their model, they can edit and run the PLaSM script as many times as needed. Each time the script is executed, the source code is sent to the server, evaluated using PLaSM, and the corresponding STL file (or an error message) is returned to the student's web browser for display.

Once the student finalizes their 3D model, they submit it for grading. At this stage, the script is sent to the server again, but this time it is accompanied by an assignment-specific Python grading script prepared by the assignment creator. The student's script is executed before the grading script. The grading script then performs three validation tests, described below, and any textual output is returned to the student's web browser for display.

If the student's solution passes all tests, a success message is displayed. Otherwise, the student receives the output generated by the grading script, along with their solution and the correct solution. Both models are displayed in the same view but differentiated by color, making it easier for the student to identify mistakes.

In the following, we take a closer look at the three validation tests performed during the grading process.

Bounding box test

The bounding box of the student's object `design` is computed and compared against the expected values specified in the assignment-specific grading script. If the values match, the object `design` passes the bounding box test. Otherwise, it fails. If this test fails, the remaining two tests are not

performed. Instead, a message is displayed in the student's web browser, providing details about the outcome of the bounding box test.

Inclusion test

This test is only performed if the bounding box test passes. On the server, PLaSM is used to generate a reference object, denoted as `interior_`, which is guaranteed to be entirely contained within the correct solution. For instance, if the correct solution is a $6 \times 6 \times 6$ cube, an appropriate choice for `interior_` would be a cube with dimensions $5.998 \times 5.998 \times 5.998$, translated by 0.001 units along each axis (X, Y, and Z) to ensure it lies entirely within the correct solution.

The inclusion test is then performed by subtracting object `design` from `interior_`. If the result of this operation is an empty set, meaning that `interior_` is entirely contained within `design`, the test passes. Otherwise, the inclusion test fails.

Complement test

This test is only performed if the inclusion test passes. On the server, PLaSM is used to generate a reference object, denoted as `exterior_`, which is guaranteed to fully enclose the correct solution as a subset. For instance, if the correct solution is a $6 \times 6 \times 6$ cube, an appropriate choice for `exterior_` would be a cube with dimensions $6.002 \times 6.002 \times 6.002$, translated by -0.001 units along each axis (X, Y, and Z) to ensure that the correct solution lies entirely within it.

The complement test is then performed by subtracting object `exterior_` from `design`. If the result of this operation is an empty set, meaning that `design` is entirely contained within `exterior_`, the test passes. Otherwise, the complement test fails.

9. Example

To illustrate the server-side grading process in more detail, assume that the student's task is to create the 3D model shown in Fig. 3. The assignment must be formulated with sufficient precision to eliminate any ambiguity regarding the shape, location, or orientation of the 3D model:

Create and display a $6 \times 6 \times 6$ *cube lying entirely in the first octant, with one vertex coinciding with the origin* $(0, 0, 0)$*. The cube should have a cylindrical*

through-hole of radius 2 *units whose axis is parallel to the X-axis and passes through the center of the cube. The resulting 3D object should be named* `cube`.

The name of the 3D object is explicitly specified as part of the assignment. While it would be possible to extract this information from the `SHOW` command in the student's script, doing so would unnecessarily complicate the grading script. For this reason, we typically do not implement such automation.

Once the student completes their 3D model and submits it for grading, the PLaSM script is sent to the server along with an assignment-specific grading script prepared by the assignment creator. Both scripts are executed sequentially on the server. The grading script consists of three validation tests, which we will examine in detail.

Bounding box test

The bounding box of any PLaSM model can be obtained using the functions `MINX`, `MAXX`, `MINY`, `MAXY`, `MINZ`, and `MAXZ`. For this specific assignment, the mathematically exact bounding box is $(0, 6) \times (0, 6) \times (0, 6)$. However, since PLaSM uses double-precision arithmetic, the computed bounding box may include small numerical inaccuracies. For example, for this particular model, the computed bounding box is `(-4.76837158203125e-07, 6.000000476837158) x (0, 6) x (-4.76837158203125e-07, 6)`.

To accommodate these minor numerical deviations, we introduce a small positive tolerance value `tol`, typically between `1e-3` and `1e-6`, depending on the model. Additionally, we round the computed bounding box values to mitigate the impact of floating-point precision issues. The following code illustrates the bounding box test for this assignment:

```
tol = 1e-3

minx = round(MINX(cube), 3)
maxx = round(MAXX(cube), 3)
if minx < 0 - tol or maxx > 6 + tol:
   print(f"In the X direction, your model spans the interval \
   ({minx}, {maxx}) but it should span the interval (0, 6).")
   return False

miny = round(MINY(cube), 3)
```

```
maxy = round(MAXY(cube), 3)
if miny < 0 - tol or maxy > 6 + tol:
   print(f"In the Y direction, your model spans the interval \
   ({miny}, {maxy}) but it should span the interval (0, 6).")
   return False

minz = round(MINZ(cube), 3)
maxz = round(MAXZ(cube), 3)
if minz < 0 - tol or maxz > 6 + tol:
   print(f"In the Z direction, your model spans the interval \
   ({minz}, {maxz}) but it should span the interval (0, 6).")
   return False
```

For instance, if the student mistakenly creates the cube in the fourth octant instead of the first, the X-axis test will pass, but the Y-axis test will fail. The resulting error message will be:

```
In the Y direction, your model spans the interval (-6, 0) but
it should span the interval (0, 6).
```

Additionally, the student's model will be displayed alongside the correct solution, as shown in Fig. 6.

Inclusion test

If the bounding box test passes, the inclusion test is performed next. This test generates an auxiliary 3D object, `interior_`, which is a subset of the correct solution. For this particular assignment, `interior_` can be created as follows:

```
interior_ = CUBE(6 - 2*tol)
MOVE(interior_, tol, tol, tol)
hole_ = CYL(2 + tol, 6)
ROTATE(hole_, 90, Y)
MOVE(hole_, 0, 3 + tol, 3 + tol)
SUBTRACT(interior_, hole_)
```

The dimensions of `interior_` are slightly reduced, and the radius of the cylindrical hole is slightly increased to ensure that `interior_` is entirely contained within the correct solution. To verify whether `interior_` is a

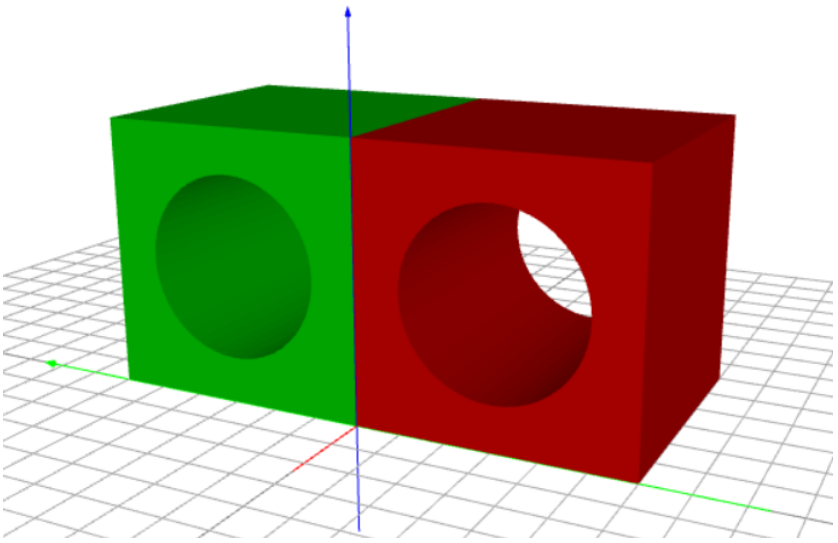

Figure 6: Student's solution (red) displayed alongside the correct solution (green).

subset of the student's solution `cube`, the former is subtracted from the latter. If the result is an empty set, the inclusion test passes; otherwise, it fails. PLaSM provides the function `EMPTYSET` for this purpose:

```
SUBTRACT(interior_, cube)
if not EMPTYSET(interior_):
    print("Your 3D model failed the inclusion test.")
    return False
```

If the inclusion test fails, the student will receive the following message:

```
Your 3D model failed the inclusion test.
```

Additionally, their 3D model will be displayed alongside the correct solution, as shown in Fig. 7.

Complement test

If the inclusion test passes, the final validation is the complement test. This test generates another auxiliary 3D object, `exterior_`, which is a superset of the correct solution. For this assignment, `exterior_` can be created as follows:

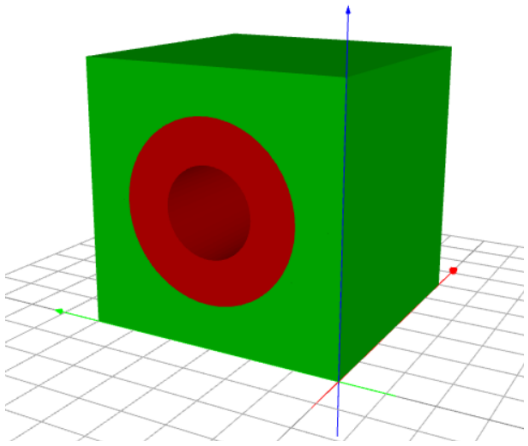

Figure 7: Student's solution (red) displayed alongside the correct solution (green).

```
exterior_ = CUBE(6 + 2*tol)
MOVE(exterior_, -tol, -tol, -tol)
hole_ = CYL(2 - tol, 6 + 2*tol)
ROTATE(hole_, 90, Y)
MOVE(hole_, 0, 3 - tol, 3 - tol)
SUBTRACT(exterior_, hole_)
```

Here, the cube's dimensions are slightly increased, and the cylindrical hole's radius is slightly reduced to ensure that `exterior_` fully contains the correct solution. To verify whether `exterior_` is a superset of the student's solution `cube`, the latter is subtracted from the former. If the result is an empty set, the complement test passes; otherwise, it fails. Since the `SUBTRACT` function modifies its first argument in place, a copy of the student's model is created before the operation:

```
cube_test_ = COPY(cube)
SUBTRACT(cube_test_, exterior_)
if not EMPTYSET(cube_test_):
    print("Your 3D model failed the complement test.")
    return False
```

If the complement test fails, the student will receive the following message:

```
Your 3D model failed the complement test.
```

Additionally, their 3D model will be displayed alongside the correct solution, as shown in Fig. 8.

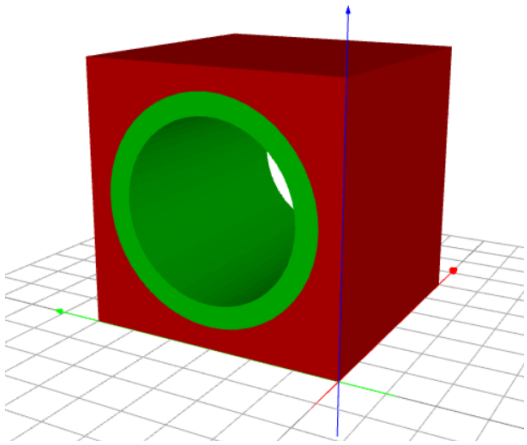

Figure 8: Student's solution (red) displayed alongside the correct solution (green).

10. Conclusions

We presented a novel self-paced, instructor-assisted approach to teaching solid modeling and CSG as an alternative to traditional classroom instruction. We discussed the advantages of using a scripting-based CAD system for learning solid modeling and CSG, compared to traditional CAD software. We explained our choice of the Programming Language of Solid Modeling (PLaSM) over other scripting CAD tools, and described the modifications we made to enhance its usability for educational purposes.

Furthermore, we detailed the implementation of automated server-side grading of student assignments, which serves as the cornerstone of the self-paced course. The structure of an introductory solid modeling course based on PLaSM is presented in Appendix A.

Appendix A. Sample 3D Modeling Course Based on PLaSM

In this introductory self-paced course, students will use PLaSM (Programming Language of Solid Modeling), a Python-based scripting CAD tool, to learn fundamental concepts in computational design, constructive solid geometry (CSG), and parametric modeling, while gaining hands-on experience in 2D and 3D modeling for engineering and design applications.

By the end of the course, students will:

- Understand the basic principles of PLaSM and its applications in computational design.
- Develop proficiency in using PLaSM commands for creating and modifying geometric models.
- Learn how to structure and optimize CAD scripts for efficiency and readability.
- Gain experience in exporting models for 3D printing and other fabrication techniques.
- Explore advanced modeling techniques including transformations, extrusions, and Boolean operations.

Section 1: Fundamental Concepts

- Introduction to PLaSM as a scripting-based CAD tool.
- Importance of using uppercase for PLaSM keywords for clarity.
- Best practices for naming objects to avoid conflicts with PLaSM keywords.
- Exporting 3D models as STL files for fabrication.
- Creating basic geometric primitives:
  - Squares with `SQUARE` (positioned in the first quadrant).
  - Rectangles with `RECTANGLE` (positioned in the first quadrant).
  - Using `BOX` to create rectangles in any location in the XY plane.
  - Circles with `CIRCLE` (centered at the origin).
- Default object colors in PLaSM and modifying them using the `COLOR` command.
- Using predefined colors (e.g., `BLUE`, `YELLOW`, `CYAN`) and material colors (e.g., `WOOD`, `COPPER`, `GOLD`).
- Displaying objects using the `SHOW` command.

Section 2: Basic 2D Modeling

- Creating fundamental 2D elements:
  - Points with `POINT`.
  - Rings with `RING`.
  - Arcs with `ARC`.
  - Triangles with `TRIANGLE`.
- Displaying multiple objects together using `SHOW`.
- Boolean operations:
  - Union of objects using `UNION`.
  - Intersection of objects using `INTERSECTION`.
  - Creating convex quadrilaterals with `QUAD`.
  - Constructing non-convex quadrilaterals by combining triangles.
  - Subtracting overlapping objects with `SUBTRACT`.

Section 3: Transition to 3D Modeling

- Creating regular polygons using the second parameter in the `CIRCLE` command.
- Extruding planar objects into 3D solids using `PRISM`.
- Constructing prisms based on various 2D shapes (e.g., squares, rectangles, polygons, circles, rings, triangles, quadrilaterals).
- Combining prisms to form complex 3D objects.
- Creating hollow objects by subtracting prisms of different heights.
- Handling overlapping surfaces to eliminate unintended color mixing.
- Erasing sections of planar and 3D objects.
- Intersecting 3D objects with planes parallel to the XY, XZ, and YZ planes.
- Constructing 3D objects based on their orthogonal views.

Section 4: Transformations and Movements

- Translating points and objects in the XY plane.
- Combining multiple translations in different directions.

Section 5: Advanced Transformations

- Splitting planar objects along lines parallel to the X and Y axes.
- Splitting 3D objects along planes parallel to the XY, XZ, and YZ planes.
- Translating objects in the Z direction.
- Reflecting planar objects about lines parallel to the X and Y axes.
- Reflecting 3D objects about planes parallel to the XY, XZ, and YZ planes.
- Scaling objects along the X, Y, and Z axes.

Section 6: Rotations and Transformations

- Rotating objects in the XY plane.
- Using positive angles for counter-clockwise rotations and negative angles for clockwise rotations.
- Selecting appropriate centers of rotation.
- Combining rotation and translation.

Section 7: Combining Transformations

- Integrating multiple transformations in the XY plane.

Section 8: Complex Modeling and Color Customization

- Applying CSG techniques learned in previous sections.
- Understanding and using RGB color codes.
- Defining custom colors and retrieving RGB values online.

Section 9: Advanced 3D Rotations

- Rotating 3D objects about the X and Y axes.
- Using the Right-Hand Rule to determine positive and negative rotations.
- Applying multiple rotations and translations to position objects correctly.

Section 10: Complex 3D Objects

- Simplifying prism creation using predefined commands (`CUBE`, `BOX`, `CYLINDER`, `TUBE`).
- Generating advanced 3D shapes such as spheres, cones, and tori.
- Constructing convex hulls from points in both 2D and 3D.
- Combining transformations and Boolean operations to develop sophisticated 3D models.

References

[1] S. Paoluzzi, Geometric Programming for Computer-Aided Design, John Wiley & Sons, 2003.

[2] R. Gobithaasan, M. Jamaludin, On teaching and learning computer aided geometric design using scientific computation software, arXiv preprint 1303.3077 (2013).
URL `https://arxiv.org/abs/1303.3077`

[3] T. Kapsalis, Cadgpt: Harnessing natural language processing for 3d modelling to enhance computer-aided design workflows, arXiv preprint (2024). arXiv:2401.05476.
URL `https://arxiv.org/abs/2401.05476`

[4] C. M. Hoffmann, Geometric and Solid Modeling: An Introduction, Morgan Kaufmann, 1989.

[5] F. Cheng, A. Joneja, Solid modeling techniques and cad/cam applications, Computer-Aided Design 27 (8) (1995) 598–614.

[6] I. D. Faux, M. J. Pratt, Computational Geometry for Design and Manufacture, Ellis Horwood Ltd, 1979.

[7] M. Mäntylä, An Introduction to Solid Modeling, Computer Science Press, 1988.

[8] A. A. G. Requicha, Representation for rigid solids: Theory, methods, and systems, ACM Computing Surveys 12 (4) (1982) 437–464.

[9] H. Chiyokura, F. Kimura, Solid modeling system with a constructive geometry capability, in: Proceedings of SIGGRAPH, 1984, pp. 147–153.

[10] C. M. Hoffmann, Solid Modeling: Shape Design and Representation, Cambridge University Press, 1985.

[11] M. E. Mortenson, Mathematics for Computer Graphics Applications, Industrial Press, 1997.

[12] J. Rossignac, H. Voelcker, Csg techniques for the design of parametric solid models, Computer-Aided Design 20 (10) (1988) 580–592.

[13] M. Mäntylä, D. G. Cameron, Parametric and Feature-Based CAD/-CAM: Concepts, Techniques, and Applications, John Wiley & Sons, 1995.

[14] D. H. Laidlaw, A. A. G. Requicha, Geometric reasoning for computer-aided design, IEEE Computer Graphics and Applications 6 (6) (1986) 36–45.

[15] I. Gibson, Efficient Solid Modeling Techniques for Computer-Aided Design, Springer, 1998.

[16] I. Fudos, C. M. Hoffmann, A graph-based approach to solving geometric constraints, ACM Transactions on Graphics 16 (2) (1997) 179–216.

[17] V. P. A. Paoluzzi, M. Vicentino, Geometric programming and solid modeling, Computer-Aided Design 35 (10) (2003) 893–903.

[18] A. S. Glassner, An Introduction to Ray Tracing, Academic Press, 1989.

[19] A. Watt, Advanced Animation and Rendering Techniques, Addison-Wesley, 2000.

[20] J. Wang, P. Benardos, Csg-based rapid design and manufacturing techniques for 3d printing, International Journal of Advanced Manufacturing Technology 42 (2009) 896–911.

[21] S. Parker, P. Shirley, Interactive ray tracing for csg objects, IEEE Transactions on Visualization and Computer Graphics 4 (3) (1998) 243–254.

[22] A. P. G. V. G. P. Van Der Laan, J. W. Hofman, Real-time boolean operations in constructive solid geometry, Computers & Graphics 38 (2014) 64–76.

[23] K. K. Chawla, Finite element methods and solid modeling in manufacturing, Journal of Materials Processing Technology 28 (1991) 177–189.

[24] J. Sarrate, X. Roca, Geometric modeling and meshing techniques for finite element analysis, Computer Methods in Applied Mechanics and Engineering 191 (2001) 1167–1184.

[25] J. J. Shah, M. Mäntylä, Parametric and feature-based cad/cam: Concepts, techniques, and applications, John Wiley & Sons (1995).

[26] P. Company, M. Contero, The need for standardized solid modeling curricula in engineering education, Engineering Design Graphics Journal 81 (3) (2017) 12–23.

[27] R. D. S. S. K. Chandrasegaran, K. Ramani, The evolution, challenges, and future of knowledge representation in product design systems, Computer-Aided Design 45 (2013) 204–228.

[28] S. A. Sorby, Educational research in developing 3d spatial skills for engineering students, International Journal of Science Education 31 (2009) 459–480.

[29] G. R. Bertoline, E. N. Wiebe, Fundamentals of Graphics Communication, McGraw-Hill, 1991.

[30] K. Lee, Principles of CAD/CAM/CAE Systems, Addison-Wesley, 2005.

[31] I. P. M. Prats, R. Company, A. Contero, Assessing the impact of feature-based modeling in cad education: A case study, Computer Applications in Engineering Education 24 (2016) 385–397.

[32] N. M. Patrikalakis, T. Maekawa, Shape Interrogation for Computer-Aided Design and Manufacturing, Springer, 2002.

[33] D. H. Laidlaw, W. B. Trumbore, J. F. Hughes, Constructive solid geometry for polyhedral objects, Computer Graphics 20 (4) (1986) 161–170.

[34] M. J. Pratt, Introduction to Solid Modeling, McGraw-Hill, 2001.

[35] K. C. J. D. Camba, An evaluation of free-form and sketch-based approaches for teaching cad modeling, Computer Applications in Engineering Education 24 (2016) 77–89.

[36] Q. Zhu, J. Guo, Virtual reality-based training for cad and solid modeling education, International Journal of Computer-Aided Engineering and Technology 11 (3) (2019) 245–261.

[37] H. Kang, J. L. Herman, Virtual reality in engineering education: A systematic review, Advances in Engineering Education 8 (2020) 1–20.

[38] A. Gomes, J. Jorge, Challenges in teaching cad: Bridging theory and practice, Computer-Aided Design and Applications 11 (5) (2014) 547–559.

[39] J. D. Camba, K. Contero, Parametric cad modeling: An analysis of strategies for learning feature-based design, Computer-Aided Design and Applications 14 (2017) 20–32.

[40] N. L. Veurink, S. A. Sorby, Enhancing visualization skills—improving performance in engineering graphics courses, Engineering Design Graphics Journal 73 (2009) 1–17.

[41] A. R. J. Morrison, Self-paced learning in engineering education: A review of research and practice, International Journal of Engineering Education 31 (2015) 545–560.

[42] F. Martin, R. Klein, Self-paced learning in engineering education: A review of its effectiveness and implementation strategies, Journal of Engineering Education 92 (2003) 355–367.

[43] A. R. B. Jaeger, M. N. Gabriele, Hands-on learning in engineering education: The role of active learning strategies, International Journal of Engineering Education 33 (2017) 812–825.

[44] R. B. S. Vajna, U. Weber, J. Zeman, A new approach to cad education: Integration of problem-solving skills and conceptual modeling, Computer-Aided Design 43 (2011) 276–290.

[45] R. Tucker, Cognitive load and its impact on learning solid modeling, Journal of Engineering Education 99 (2) (2010) 185–198.

[46] I. Gibson, Advanced Manufacturing and Solid Freeform Fabrication, Butterworth-Heinemann, 1998.

[47] Z. Shen, R. M. Grisso, Teaching algorithmic modeling for engineering design: A case study in cad education, Journal of Engineering Design 23 (2012) 35–52.

[48] G. S. D. Perkins, Teaching algorithmic thinking in engineering: A cognitive perspective, International Journal of Technology and Design Education 28 (2018) 57–72.

[49] A. C. J. D. Camba, K. Contero, Automated assessment of parametric cad models using feature-based comparison, International Journal of Computer Integrated Manufacturing 33 (2020) 1–18.

[50] NCLab, Nclab: Learning for industry 4.0, `http://nclab.com`, accessed: 2025-02-08 (2025).

[51] Digital Learning Institute, What is self-paced learning? definition, benefits, and tips, accessed: 2025-02-08 (2024).
URL `https://www.digitallearninginstitute.com/blog/what-is-self-paced-learning-definition-benefits-and-tips`

[52] S. Karatas, H. Arpaci, The effect of self-directed learning readiness on student achievement in online learning environments, Turkish Online Journal of Educational Technology 14 (2015) 77–87.

[53] N. Zacharis, Theory of planned behavior and online learning: A self-regulation perspective, Computers & Education 56 (2011) 1034–1044.

[54] N. A. Bradbury, Attention span during lectures: How long is too long?, Advances in Physiology Education 40 (2016) 509–513.

[55] E. A. F. D. M. Bunce, K. Y. Neiles, How long can students pay attention in class? a study of student attention decline using clickers, Journal of Chemical Education 87 (2010) 1438–1443.

[56] Academy of Mine, The benefits of self-paced learning, accessed: 2025-02-08 (2024).
URL https://www.academyofmine.com/self-paced-learning-benefits

[57] M. C. L. T. de Jong, Z. C. Zacharia, Physical and virtual laboratories in science and engineering education, Science 328 (2010) 889–893.

[58] Creatrix Campus, 10 benefits of self-paced learning, accessed: 2025-02-08 (2024).
URL https://www.creatrixcampus.com/blog/10-benefits-of-self-paced-learning

[59] B. J. Zimmerman, Becoming a self-regulated learner: An overview, Theory Into Practice 41 (2002) 64–70.

[60] P. R. Pintrich, A conceptual framework for assessing motivation and self-regulated learning in college students, Educational Psychology Review 16 (2004) 385–407.

[61] P. Solin, Self-paced, instructor-assisted approach to teaching linear algebra, Mathematics in Computer Science 15 (4) (2021). doi:DOI:10.1007/s11786-021-00499-z.

[62] P. Solin, A. Freyer, Self-paced, instructor-assisted approach to teaching python programming, Mathematics in Computer Science 17 (2) (2023). doi:DOI:10.1007/s11786-023-00560-z.

[63] P. Solin, Self-paced, instructor-assisted approach to teaching SQL, Journal of Computational and Applied Mathematics (2024). doi:submitted.

[64] P. Koenig, M. Knecht, Teaching parametric modeling with openscad in engineering education, International Journal of Engineering Education 31 (2) (2015) 573–582.

[65] S. Grover, R. Pea, Computational thinking in k-12: A review of the state of the field, Educational Researcher 42 (2013) 38–43.

[66] J. Smith, J. Doe, The impact of programming on attention to detail and analytical thinking, Journal of Educational Computing Research 58 (3) (2020) 567–588.

[67] R. Jones, Debugging and attention to detail: A study on problem-solving in coding, Computer Science Education Review 35 (2) (2018) 245–260.

[68] T. A. Y. Liao, M. Hsu, Developing critical thinking and attention to detail through computational programming tasks, International Journal of STEM Education 6 (2019) 21–38.

[69] A. McKenna, C. Froyd, Student strategies for navigating cad interfaces: Implications for design education, International Journal of Engineering Education 34 (2018) 367–378.

[70] P. C. G. Ameta, J. D. Camba, Assessing student design thinking in cad: The role of explicit vs. implicit modeling strategies, Computer-Aided Design and Applications 10 (2013) 345–356.

[71] X. Chen, R. Smith, Automated assessment in cad learning environments: A framework for real-time feedback, International Journal of Technology and Design Education 28 (2018) 487–506.

[72] B. Bickel, G. Noris, B. Thomaszewski, Computational design and scripting in engineering education, Computer-Aided Design 102 (2018) 105–118.

[73] E. Stoyanov, R. Angelova, Integrating algorithmic thinking and cad scripting for engineering education, Journal of Computer-Aided Design and Applications 18 (4) (2021) 521–536.

[74] J. Fowler, B. Bender, Teaching computational thinking through cad modeling and parametric design, Computer Applications in Engineering Education 25 (2017) 845–862.

[75] T. U. M. Weiler, D. Seidel, Computational thinking in cad education: Analyzing student approaches to algorithmic design, International Journal of Computer Science Education 30 (2020) 95–112.

[76] S. Paoluzzi, Functional programming in computational geometry and design, Journal of Computer-Aided Design 29 (1) (1997) 19–34.

[77] G. Cervino, A. Paoluzzi, An introduction to functional programming for geometric computing, Journal of Symbolic Computation 27 (1999) 355–380.

[78] L. R. S. A. Paoluzzi, M. Vicentino, Parametric and procedural modeling with plasm, Computer-Aided Design and Applications 2 (2005) 361–372.

[79] U. Cugini, C. Rizzi, N-dimensional geometric modeling with plasm, Computer-Aided Geometric Design 17 (2) (2000) 87–102.

[80] G. C. R. Maffei, A. Paoluzzi, Topological data structures in solid modeling and their applications, Engineering with Computers 18 (2002) 102–118.

[81] M. Fontana, Interfacing plasm with finite element analysis tools, Engineering Computations 18 (4) (2001) 311–329.

[82] U. Cugini, A. Paoluzzi, Teaching computational geometry and solid modeling with plasm, Computer Applications in Engineering Education 10 (2002) 127–140.

[83] S. Paoluzzi, F. Scorzelli, Parametric architectural design using plasm, Automation in Construction 19 (5) (2010) 610–621.

[84] A. Fioravanti, Teaching computational design through plasm, International Journal of Architectural Computing 2 (2) (2004) 205–222.

[85] F. Guerra, M. Mauri, Using plasm in cnc machining and digital fabrication, Journal of Manufacturing Systems 32 (3) (2013) 450–461.

[86] G. Di Blasi, Scientific visualization using functional geometric modeling, Journal of Visualization 19 (4) (2016) 523–538.

[87] M. V. A. Paoluzzi, G. Cervino, Functional geometric modeling for scientific data analysis, International Journal of Computational Science and Engineering 7 (2012) 143–162.

[88] A. Paoluzzi, F. Polverini, Teaching computational design with functional cad tools, Education and Information Technologies 12 (2007) 45–61.

[89] M. Attene, A. Paoluzzi, Geometric reasoning and algorithmic modeling in engineering education, Computer-Aided Design and Applications 10 (2013) 525–538.

[90] PLaSM (procedural wrapper), `https://github.com/femhub/pyplasm` (Accessed February 10, 2025).

[91] P. Solin, Introduction to 3D Modeling, A Project-Based Approach, NCLab, 2018.