AUTOMATED GRADING OF PARAMETRIC MODELLING ASSIGNMENTS

A Spatial Computational Thinking Course

DEREK PUNG[1], DO PHUONG TUNG BUI[2] and
PATRICK JANSSEN[3]
[1,2,3]*National University of Singapore*
[1]*derek.pung@u.nus.edu* [2]*phuongtung1231992@yahoo.com*
[3]*patrick@janssen.name*

Abstract. This paper describes the implementation and deployment of an automated grader used to facilitate the teaching of a spatial computational thinking course on the online education platform, edX. Over the period of a course on the platform, more than 3000 assignments were graded. As an evaluation of the grader, examples of assignments and statistical results are presented and discussed.

Keywords. Automated Assessment; Parametric Modelling; MOOC.

1. Introduction

The rise of the web provides people new ways to interact and learn. In recent years, Massive Open Online Courses (MOOCs) have allowed students to receive a quality education through the web. Some institutes have since taken up a “flipped classroom” approach in their pedagogy, where curricular content was taught outside the formal system and in affiliated online systems. Industry professionals alike make use of such platforms to keep themselves updated with the relevant skills. In the spatial discipline, Spatial Computational Thinking is increasingly being recognised as a fundamental skill. This paper describes the application of an automated online grader used to facilitate the teaching of Spatial Computation principles on a MOOC.

The use of a web-based grader is common in online learning platforms for learning programming. Numerous studies have highlighted the advantages of automated formative assessment (Lewis & Davies 2004, Douce et al. 2005, Nicol & Macfarlane-Dick 2007, Baranaa & Marchisioa 2016). Automated formative assessment provides a system where students can conveniently test their understanding of taught concepts, which is synchronous with the pedagogical objectives of a MOOC.

Approaches for assessing programming assignments may be summarised into two categories: dynamic analysis and static analysis (Ala-Mutka 2005). A submitted file is executed in a dynamically assessed assignment and not executed in a statically assessed assignment. The former checks for functionality and efficiency of the code while the latter checks for programming styles and design. Numerous systems for automating assessment have been developed for learning textual programming languages. One of the earliest systems was the AGSICP for Cobol Programming (Aaronson 1973). More recently, two popular systems are Web-Cat and Stepic.

For parametric modelling, no automated assessment approaches have been developed. In general, the approach could be similar to existing approaches using automated assessment for learning textual programming languages. However, for parametric modelling, the output of the program may be a complex 3D model. A more advanced approach is therefore needed for assessing the validity of such models.

The paper describes the development of an automated grader for parametric modelling assignments. The contents of this paper are organised as follows. Section 2 sets the premise the grader was designed for. Section 3 describes our implementation of a grader for parametric modelling assignments. Section 4 provides an example of how the grader is used in a MOOC. Finally, Section 5 concludes the paper.

2. Context

The Automated Grader was used in a second-year module, “Spatial Computational Thinking” at the National University of Singapore. The module consisted of 150 students, most of whom did not have prior scripting or programming knowledge. The module focused on the development of algorithms for generating complex, parametric 3D models. It was taught on the edX MOOC platform using videos and online exercises. The modelling assessments were all performed in the web-based parametric modelling tool developed by the authors (reference removed). The Automated Grader was developed on the Amazon Web Services (AWS) infrastructure, allowing large numbers of assignments to be graded in parallel.

For each assignment, the instructors created a detailed problem description on the edX platform, and a model representing the correct answer was uploaded to the Amazon Simple Storage System (S3) object storage. Students then performed the modelling assignment and uploaded their models through the edX platform. In the edX platform, the student models were added to a grading queue, from which the AWS server would fetch the models and grade them using an AWS Lambda function, returning the grade and any feedback to the edX server. Within a maximum of 20 to 30 seconds, students would see the grade and feedback displayed on their browser.

The modelling assignments increased in complexity as the semester progressed. At the start of the semester, the assignments started quite simple. As the students learnt new concepts and techniques, the assignments also grew in complexity.

The automated grading process meant that these assignments could not be open-ended. Each student was expected to submit a model that, given certain inputs (such as parameter values), produced ‘correct’ outputs. The way that the models were implemented could still differ from one student to the next. However, the outputs were required to be the same for all students. This, of course, meant that individual creative freedom was quite limited. In order to overcome this limitation, the final assignment of the semester was open-ended and was manually graded. Nevertheless, it is noted that automated grading freed up a significant amount of time that allowed instructors to spend more time helping students with their final assignments.

3. 3D Model Grader

The grader was designed to be an extension of a web-based parametric modelling tool developed by the authors. A similar approach could be developed for other existing parametric tools. However, for commercial software issues with commercial licenses would have to be resolved if parallel execution of the grader would be required.

Each parametric model has a set of input parameters. Setting the values for these parameters and executing the model will result in a 3D geometric output, which we refer to as the *output model*. Each assignment was accompanied by an answer model and a number of input parameter sets. The grading process then consisted of the following steps:

- The parameters in the parametric answer model were compared to the parametric submitted model. If the submitted model had missing parameters, the grader would exit and assign a zero grade.

For each set of input parameters:

- The parametric answer model and parametric submitted model were both executed, thereby producing two output models: the submitted output model and the answer output model. These two output models were compared to one another, and a grade was calculated.

3.1. MODEL NORMALIZATION

One complication that had to be addressed was the normalization of the output models. The ordering of the geometric objects and entities in an output model should not impact the correctness of the model. For example, the answer output model may contain two polygons. If the submitted output model contains the same two polygons, it should be marked as correct, irrespective of the polygon ordering. The same applies to the ordering of other geometric entities, including vertices. Thus, in order to be able to compare models, they first needed to be normalized, so that the model representation could be guaranteed to be deterministic.

The normalization process consisted of two stages. First, the vertex order within individual objects was normalized. Second, the order of the objects in the model was normalized.

For the normalization of vertex ordering, a few different rules were applied. For open polylines, they were modified in order to guarantee that the coordinate values for the start vertex were always less than the end vertex. For polygons and closed polylines, they were modified in order to guarantee that the first vertex was always the vertex with the lowest coordinate values. For polygons with holes, additional rules developed.

For the normalisation of object ordering, a *fingerprint* was generated for each object in the model. The fingerprints were generated from the data constituting that object, including the coordinates of the vertices. (For the fingerprinting process, all numerical values were rounded to 8 significant digits. This accounted for rounding errors that could result from different computing hardware and operating systems.) For entities that were not exact clones, each fingerprint was guaranteed to differ. The normalisation process then sorted all entities according to their fingerprints.

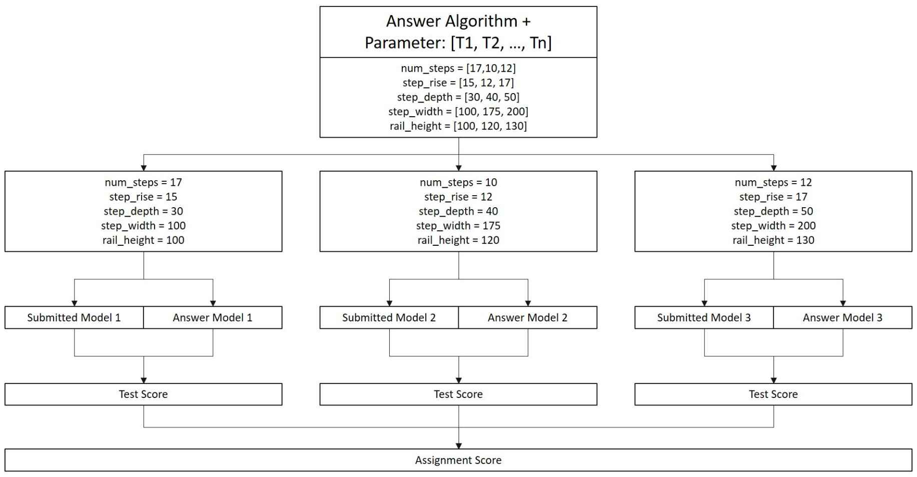


Figure 1. Schematic flow of information in an assessment with five parameters and three tests.

3.2. MODEL SCORING

In each test, the object (points, polyline, and polygons) in the answer output model were matched against the objects in the submitted output model. If a match was found, then the score was incremented by 1. The matching process required an exact match between objects, using the same fingerprinting process as was used for model normalization. To encourage efficient solutions, extra objects found in the submitted output model result in 1 mark being deducted (equation 1).

$$E_{type} = (n_{congruent} - a) \tag{1}$$

where:			
	E_{type}	=	Score for Entity Type (point, polygon, or polyline)
	$n_{congruent}$	=	Number of entities (of type) generated in the submitted model which are congruent to the ones generated in the answer model
	a	=	1 or 0; 1 mark is deducted if extra entities were created for an entity type

Figure 2. equation 1.

$$S_{test} = (\sum E_{type} + A_{equal})/(N_{entities} + N_{attributes}) \tag{2}$$

where:			
	S_{test}	=	Score for a test
	E_{type}	=	Score determined by equation (1)
	A_{equal}	=	Number of attributes in the submitted model which are equal to the ones in the answer model
	$N_{entities}$	=	Number of entities in the answer model
	$N_{attributes}$	=	Number of attributes in the answer model

Figure 3. equation 2.

$$S_{final} = \bar{S}_{test} \tag{3}$$

Figure 4. equation 3.

The score awarded to each test is the fractional result from the sum of all the number of congruent points, polylines, polygons, and model attributes in the submitted model against the total number of entities and attributes in the answer model (equation 2). The final grade given to an assessment is determined by the average of the test scores (equation 3). For example, consider an answer generated model has 10 polygons. If only 8 of those polygons can be matched against entities in the submission generated model, then the grade would be 8/10, or 0.8. An example breakdown of the grading is detailed in Section 4.1.

3.3. GRADER FEEDBACK

Feedback messages highlight to the students how marks were lost and allow them to rectify and receive a better grade in their subsequent tries. Such information returned from an automated assessment is essential in the facilitation of a self-paced course. The key feedback messages are listed as follows:

- Entities could not be found
- Extra Entities
- Model Attribute not found

- Incorrect model attribute value
- Missing start node parameters

In the web parametric modeller the authors have created, entities may be assigned with attributes. Attributes define the types of data that may be attached to entities in the entire model. They exist in key-value pairs in which the user can specify data to be stored under a name key. Materials are assigned to entities through the use of attributes. The materials of a submitted model may also be checked.

A submitted model with missing entities translates to a parametric model with wrong procedures. A submitted model with extra entities would be another with redundant procedures. In addition, the grader was able to pick up translational and directional errors in the geometries which would be highlighted to the student in the returned feedback.

4. Spatial Computational Thinking

The learning outcome of the course was to gain theoretical knowledge and practical skills in applying spatial computational thinking as a way of generating 3D models, building upon elementary critical and logical thinking aptitude. The key concepts tested and the weekly assignments are listed as follows:

- W1 - Variables and Operators: Hello World (Console-Based).
- W2 - Lists, Control Flow, and Functions: Printed Hash Checkerboard (Console-Based)
- W3 - Entities and Attributes: Debugging Generated Geometry (Static Geometry)
- W4 - (break)
- W5 - Rendering and Geometric Constraints: Windows (Parametric Geometry)
- W6 - Search Spaces, Vectors, and Planes: Parametric Stairs (Parametric Geometry)
- W7 - Loop Updates and Transformations: Parametric Stair Runs (Parametric Geometry)
- W8 - Vector Arithmetic and Graphing Polylines: Solar-Responsive Roof (Parametric Geometry)

To familiarise the students with procedural thinking, the earlier assignments were simple 1-part tasks. In the later weeks when more complex concepts were introduced, each assignment was broken up into smaller steps to guide the students into creating the final model while building up their capacity to deconstruct a spatial problem.

The assignment for Week 1 was a simple 1-part task that required the students to submit a file with a single parameter. The console should print "Hello __parameter_value__" when executed. In Week 2, the students were given another 1-part task which required them to submit a file with a single parameter that defined the size of a printed checkerboard. The students were first introduced to geometry in Week 3. For their assignment, they were given a file that generated an extruded model and they were tasked with changing the arguments and values in the procedures to achieve another extruded result.

The assignment for Week 5 was a 3-part task that broke the creation of a window into the creation of a polygon, the creation of a grid of panes, and finally the extrusion of the frame. In Week 6, the students created a flight of stairs. They were first tasked with creating a staggered polyline that was then translated to the profile of the stairs. Next, they were required to create the volume of the stairs. Finally, the railings are created. This 4-part task will be discussed in detail in Section 4.1. The students were tasked with another stairs assignment in Week 7. In this 4-part assignment, the students first had to create a run of steps on a plane. Next, the last step was modified into a landing. Then, they were required to create runs of steps one after another. The model was required to be able to change direction based on a parameter. Finally, the number of steps in each run was made variable by a parameter. The final assignment in Week 8 looked at the creation of a parametric solar roof that reacted to the direction of the incoming sun rays. The assignment was broken down into three parts. The first required the students to create a series of arches. Next, the students created a barrel vault. Finally, glass panels with varied sizes were installed on the vault. Panels with direct exposure to the incoming rays were smaller in size.

4.1. PARAMETRIC STAIR WITH RAILING

The parametric stair model served as an introductory modelling task to more advanced concepts like transformations with vectors and planes. As an example of a typical submission to the automated grader, the task featured in the course has been chosen to be described in detail.

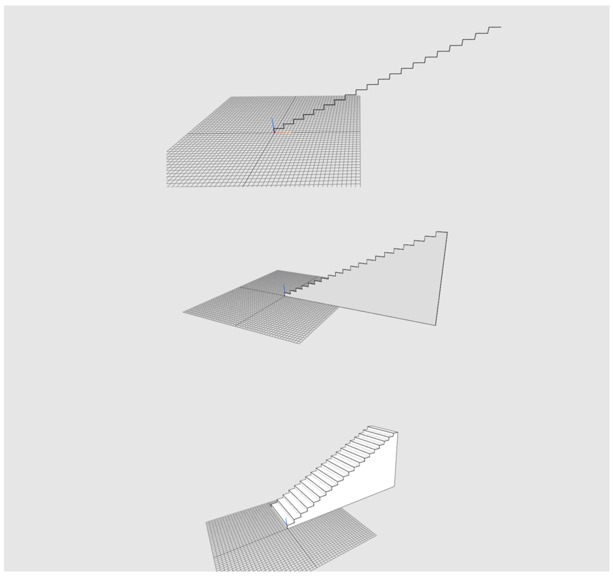
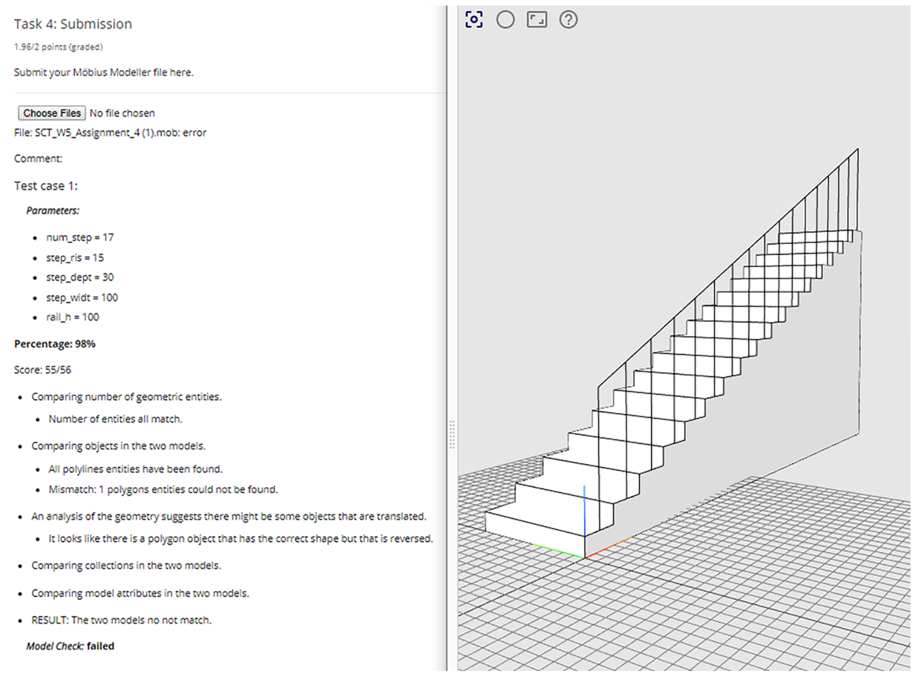


Figure 5. The 4 parts of the assignment (Tasks 1 to 3). Example of an erroneous submission in Task 4.



Task 4: Submission

1.96/2 points (graded)

Submit your Möbius Modeller file here.

Choose Files No file chosen

File: SCT_W5_Assignment_4 (1).mob: error

Comment:

Test case 1:

Parameters:

- num_step = 17
- step_ris = 15
- step_dept = 30
- step_widt = 100
- rail_h = 100

Percentage: 98%

Score: 55/56

- Comparing number of geometric entities.
  - Number of entities all match.
- Comparing objects in the two models.
  - All polylines entities have been found.
  - Mismatch: 1 polygons entities could not be found.
- An analysis of the geometry suggests there might be some objects that are translated.
  - It looks like there is a polygon object that has the correct shape but that is reversed.
- Comparing collections in the two models.
- Comparing model attributes in the two models.
- RESULT: The two models no not match.

Model Check: **failed**

Figure 6. The 4 parts of the assignment (Task 4). An example of an erroneous submission.

This assignment was divided into four parts (Figure 5 and 6). First, the students were tasked to create a staggered polyline which would translate to the profile of the stair. Step depth, height, and number were set as parameters. Next, the cross-section polygon was extruded in the direction of its normal based on another parameter. Finally, the polyline rails were added. The height of the rails was also defined by another parameter.

Following the submission through edX, the model was run through three tests and each generated model was positionally compared to the one generated by the answer algorithm (Figure 1). The fractional score calculated from the average of the three tests was returned to edX, where the final grade was converted into the edX weighted grade.

Figure 6 also shows the most commonly detected error for the task. A polygon with a reversed list of positions was determined to have its normal pointed in the opposite direction and was treated as a different polygon. As seen in the figure, the face opposite to its normal is rendered with a darker shade in the viewer. With 1 polygon missing in a model with 56 entities, a score of 55/56, or 0.98, was given for the first test. Similarly, the submission generated model was scored 34/35, and 40/41 in its next two tests respectively. The task was awarded a final score of 0.98.

4.2. RESULTS

Amazon CloudWatch was set up to log all submission processes. Over the period of the course, over 3000 submissions were registered. The authors used the collected information to make incremental improvements to the grader feedback. Negative scoring of extra entities was introduced in Week 6, along with detailed suggestions for missing entities. A gradual decrease in the number of submissions with extra entities over the subsequent weeks can be seen in the data (Figure 6). This suggests an improved quality in the submissions.

	w5	w6	w7	w8	total
entities could not be found	144	192	224	209	769
reversed	-	66	20	15	-
translated	-	3	93	97	-
extra entities	0	199	148	88	435
model attribute not found	28	7	0	18	53
wrong model attribute value	25	3	0	7	35
missing start node parameters	18	24	40	29	111
Total Erroneous Submissions	215	425	412	351	1403
Total Submissions	528	857	846	920	3151

Figure 7. Frequency of Errors for Parametric Geometry Assignments.

In general, the automated grader achieved the desired results. It gave students immediate feedback on their parametric modelling assignments, allowing them to learn at their own pace. However, feedback from students also highlighted that they often did not understand why a model was being marked as incorrect. For example, for the common error of the reversed polygon mentioned above, the grader only reports that one of the polygons is incorrect. It does not explain why it is incorrect.

In order to tackle this issue, a set of additional checking algorithms were implemented to detect common errors. For example, if a student model was missing a polygon, the algorithm would check if the reversed polygon was present. If it was present, it could then feedback to the student that the polygon was reversed. Many other checks were performed, for example, polygons that were correct in shape and orientation, but that were placed in the wrong location (often due to some small offset). However, as models became larger, this checking process got increasingly complex and slow to execute. In the end, the decision was taken to reverted back to simpler checks that were faster to execute. Nevertheless, in order to achieve the pedagogical objectives, it is important that students are able to understand why their models are being graded as incorrect. Our investigations into this are ongoing.

5. Conclusions

Our ongoing research aims to investigate the extent to which the visual programming language can support computational thinking concepts required for constructing complex parametric models. This research paper focuses on an extension of our web-based parametric modeller for automated assessment of parametric modelling assignments, and a framework using modern cloud technologies which may be replicated on other web-based parametric modelling systems.

The grader is being further developed and improved. Three key areas are being worked on. First, the grader currently only gives textual feedback to the students. We are investigating approaches whereby the grader might be capable of giving visual feedback. Second, the grader currently requires all students to be given the same problem, which causes problems with plagiarism. We are investigating approaches whereby questions can be randomized so that each student will receive a variant of the question, which will require a slightly different answer. Third, the grader is currently only grading the geometric output model that is generated. We are investigating how the grader could support performative grading. For example, the assignment may be to create a space with certain daylight and solar radiation requirements. In such a case, students will be able to submit different models, and as long as the model meets the performative requirements, the model will be marked as correct.

References

Aaronson, N.M.: 1973, AGSICP, An automated grading system for the instruction of Cobol programming, *ACM 1973.*, 428.1-429.

Ala-Mutka, K.M.: 2005, A Survey of Automated Assessment Approaches for Programming Assignments, *Computer Science Education*, **15**(2), 83-102.

Barana, A. and Marchisio, M.: 2016, Ten Good Reasons to Adopt an Automated Formative Assessment Model for Learning and Teaching Mathematics and Scientific Disciplines, *Procedia - Social and Behavioral Sciences*, **228**, 608-613.

Douce, C., Livingstone, D. and Orwell, J.: 2005, Automatic test-based assessment of programming, *Journal on Educational Resources in Computing*, **5**(3).

Lewis, S. and Davies, P.: 2004, Automated peer-assisted assessment of programming skills, *ITRE 2004. 2nd International Conference Information Technology: Research and Education*, 157-166.

Nicol, D.J. and Macfarlane-Dick, D.: 2006, Formative assessment and self□regulated learning: a model and seven principles of good feedback practice, *Studies in Higher Education*, **31**(2), 199-218.