EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling (2008)
C. Alvarado and M.- P. Cani (Editors)

Using Sketches and Retrieval to Create LEGO Models

Tiago Santos Alfredo Ferreira Filipe Dias Manuel J. Fonseca

Department of Computer Science and Engineering
INESC-ID/IST/Technical University of Lisbon
R. Alves Redol, 9, 1000-029 Lisboa, Portugal

Abstract

In this paper we describe a system to create LEGO® models using sketches. Although there are a few applications to create LEGO models, they are difficult to use, mainly due to the searching and manipulation mechanisms that they (do not) offer. Here, we propose a sketch based approach, where users can easily insert parts, by specifying their dimensions through sketches and the system suggests a list of possible parts. To help with the modeling and the manipulation we also developed a constraint based mechanism, which keeps parts connected, performs snap-to-grid and detects collisions. Experimental tests with users revealed that our approach is easier and faster to use than a conventional application, such as LeoCAD.

Categories and Subject Descriptors (according to ACM CCS): H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval H.5.2 [Information Interfaces and Presentation]: User Interfaces

1. Introduction

The construction of LEGO models is still in the memories and in the hobbies of many people around the world. Nowadays, besides the construction of real models, we can also build virtual models using specific LEGO applications, such as, the MLCAD [Lac], LeoCAD [leo] and LEGO Digital Designer® [ldd].

Although these tools allow the creation of different types of LEGO constructions, they fail to mimic the physical interaction between parts, like collisions and connections. Additionally, they do not provide any efficient search/retrieval mechanisms to help users find parts in large collections.

To overcome these problems, we developed a system, which combines calligraphic interaction, a constraint solver and a retrieval mechanism. Our solution uses sketches to specify the dimensions of the desired part, to control the camera and to perform actions on existing elements.

The retrieval component uses the dimension or dimensions of the part, specified through sketches, to perform a search into the database. Results that satisfy the specifications are presented to users in a suggestion list (see list on the right of Figure 1). After selecting the desired part from the list, users can manipulate it as they do with real parts. This is possible due to our constraint solver and snap-to-grid

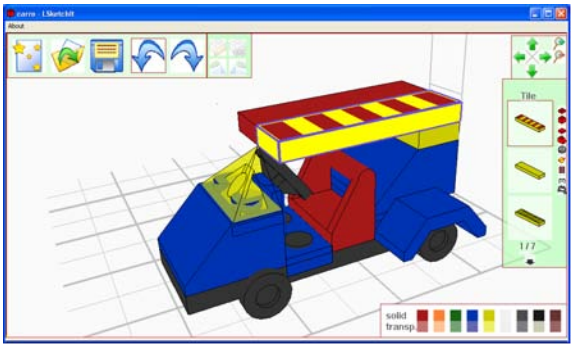

Figure 1: *General overview of the LSketchIt application, showing the suggestion list on right and the color palette on the bottom.*

mechanisms. Our system also detects if parts are above others, and in this case it creates connections. Additionally, it detects collisions and moves parts to the top of others, when users force the collision. This collision and constrain modules also know how to deal with connected parts, moving them all together. Moreover, it has the concept of gravity, preventing parts to remain in the air when users delete blocks that are below them, to mimic real constructions. Finally, we

added a snap-to-grid constraint to easier the positioning of parts and to reduce the construction time.

Experimental evaluation with 10 users showed that users can make more complex creations with our solution than with LeoCAD, mainly due to the searching and manipulation facilities provided by our system.

The rest of the paper is organized as follows. In the next section we describe some existing applications for creation of virtual LEGO models and we analyze some constraint based systems. Section 3 gives an overview of our solution, while section 4 describes the retrieval mechanism. In section 5 we describe the construction process and how parts are combined. Section 6 describes the calligraphic interface, while in section 7 we present the results and analysis of the tests performed with users. In the last section we present conclusions and discuss future work.

2. Related Work

As we identified previously, there are some solutions to allow the creation of LEGO virtual models. In this section we shortly describe the most used applications to this end. Additionally, and to complement the knowledge needed to develop a more complete solution, we analyzed some constraint-based 3D modeling systems, identifying possible solutions to integrate into our approach.

2.1. LEGO Modeling Systems

During our research on existing applications for LEGO creation, we identified three main tools: the MLCAD [Lac], the LeoCAD [leo] and the LEGO Digital Designer [ldd]. The first two applications are open-source projects, with a large user community, and they use the LDraw open-source library to represent parts in 3D. The other tool is a proprietary system from the LEGO company.

The LEGO Digital Designer is an application, based on the WIMP paradigm. It is a 3D space modeler with snap-to-grid and connections between parts. The main drawbacks of this application are the searching mechanism and the manipulation of the camera.

The MLCAD is a very complete and also very complex system for experts, where the main advantage lays in the background community that supports it. Although, it is a very complete application with a large creativity freedom, it is almost impossible to edit the model in 3D or select a part from the database. Additionally, it does not have any constraint like connection between parts nor collision detection.

LeoCAD, like MLCAD, is also based on the LDraw library. It is a simple application, with little functionality, but it allows users to model LEGO creations in 3D. The part and camera manipulation are very difficult and unnatural, because users have a lot of freedom and are not guided to complete the operation. Moreover, this system does not have any constraint, collision or connection mechanism to be applied to parts. The big advantage of LeoCAD is in the fact that it already includes a library of parts, translated from the LDraw library. This way, users do not have to install the LDraw library, as they have in the case of the MLCAD.

More recently Baradaran presented a comparative evaluation between real LEGO bricks and a virtual LEGO software [BS06]. Authors created two user interfaces, one controlled by a 2D mouse, and other by a 3D input device with force-feedback. Experimental results showed that using real LEGO parts is significantly faster for first-time users than using virtual LEGO software. Although this study is interesting, it focus only on manipulation, since parts are already placed in the model and users never have to search for them.

2.2. Constraint-based 3D modeling

A lot of work has been done in the past years in the field of modeling with constraints. The first system introducing the concept of gestures and constraints was Sketchpad [Sut63], which allowed users to specify constraints between objects, such as, parallelism, perpendicularity, etc.

Zeleznik et al. [ZHH96], developed SKETCH, a gesture-based interface to create 3D primitives, such as cubes, cylinders and pyramids. Geometric transformations, translations and rotations are conditioned by constraints. The dragging allows users to pick an object and drag it freely or according to a line drawn by the user.

Google SketchUP [ske] has a friendly and simple interface with good help tips. It uses construction lines that are familiar to users. The constraints are presented via suggestions, using snapping techniques. For example: interceptions, parallelism, perpendicularity, medium points, etc.

Some applications use suggestive interfaces, through the use of expectation lists, which allow users to resolve ambiguous situations. Chateau [IH01], by Igarashi, is a system that combines a gesture interface with the suggestions from Pegasus [IMKT97]. Chateau provides hints to users, in a tentative to decipher the intentions of the user by analyzing the surrounding context. Snapping and prediction is used to solve some constraints.

Another system that uses a suggestive interface is Gides++ [PBJ*04]. This tool uses the paper and pencil metaphor and is suitable for the initial stages of the design process. It uses a large variety of constraints, combined with an incremental drawing paradigm and a suggestion mechanism. Users can also use construction lines to help or constraint the modeling task.

The CIGRO [CNJC03] uses only sketches to create 3D models. It is dedicated manly to the preliminary stages of product design. To help users in their modeling task, the

system uses auxiliary (or construction) lines, which serve as skeleton of the model. On top of these lines users refine the model by pressing the pen harder. It has a set of constraints, like parallelism perpendicularity, proximity, as well as a set of gestures to edit. This system is limited to lines and axonometric projection.

Zou and Lee describe in [ZL07] an approach that uses a "beautification" mechanism to reconstruct 3D models from inaccurate 2D sketches. Their method detects geometric constraints, such as parallel and orthogonal faces and then perform a selection of a subset of constraints depending of their type.

Shin and Igarashi presented recently a system for quick construction of 3D scenes composed of multiple objects [SI07]. Their system takes simple 2D sketches of models as input. Then, it automatically identifies corresponding models in a database and puts them in the appropriate location and posture, matching the user's input sketches. Their system, combines a 3D search mechanism with a 3D posture estimation technique (constraints) to obtain the desired result.

There are also commercial modeling applications that use proprietary constraint solvers, such as 3D Studio Max or Maya. These give users complete control over the creation, leading to a very painful task that requires a lot of visits to the help. Another major disadvantage is that the 3D models are, basically, individual objects.

3. System Overview

As we have seen previously, there are several studies on the use of constraints in 3D modeling. In this paper, we describe the integration of constraints, calligraphic interfaces and retrieval mechanisms, to simplify and speed up the creation of LEGO models.

Figure 2 presents an overview of the different components of our system, called LSketchIt.

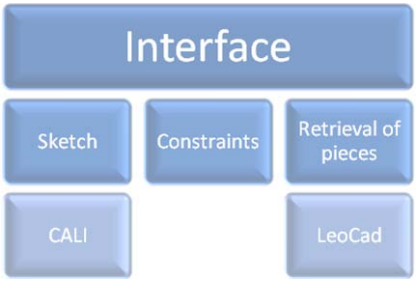

Figure 2: *System overview.*

The main components of our system are the constraint and the retrieval modules. In the retrieval part, we implemented a mechanism to search parts according to various characteristics, such as width, height, etc., trying to improve one of the major problems in the current applications, the location of specific parts.

To improve the combination/construction of LEGO models, we implemented a module that deals with the constraints between parts. This module is responsible to create the connections between parts, to propagate the transformations (translation and rotations) and to maintain the reality (gravity) when a part is deleted or moved.

The calligraphic interface connects the two previous modules, and gives a simple way for users to specify the part to search for. Additionally, we also implemented some commands to allow the manipulation, construction and visualization of LEGO models.

4. Retrieving LEGO Parts

In this section we describe how the retrieval mechanism for LEGO parts was developed and how it works.

4.1. Part Library

The LEGO parts belong to the LEGO Company and they are not freely available. To overcome this, we searched for existing open-source libraries and found the LeoCAD library. We chose this because of the information that it has associated to each part, namely, name, dimensions and category, which are very useful for our retrieval mechanism. The original LeoCAD library started from the set of LEGO parts and created 32 categories. For our work, we analyzed these categories and managed to reduce them to nine main groups (Plates, Bricks, Tile, Slope Brick, Technic, Space, Train, Other Bricks and Accessories).

4.2. Search Mechanism

Our search mechanism relies mainly on the information about dimensions and categories of parts that are stored in the library. When users want to search for a part, they can specify only one dimension (e.g. width, length or height) or more than one dimension (e.g. width and length), while modeling (see Figure 3). Then, our retrieval system compares dimensions defined using sketches with the part dimensions stored in the pattern "width x length x height".

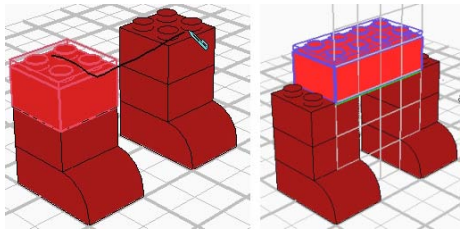

Figure 3: *Searching a part by specifying its width and length.*

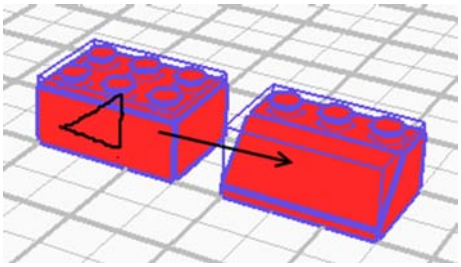

Figure 4: *Refinement of the results by sketching (a triangle) over an existing part.*

When this information is not associated to the part, we use the dimensions of its bounding box.

To enrich the retrieval process, we allow the over sketching of gestures on parts to refine the search and find specific types of parts. For instance, if we draw a triangle inside the retrieved or selected brick, the system will refine the query, showing only bricks with slopes (see Figure 4). Table 1 shows the different combinations of gestures and the produced result.

Gesture	Outside part	Inside part
Line		Pin
Circle	Tyre, Wheel, Round	Side, Hole
Rectangle	Baseplate, Plate, Brick	
Triangle	Slope Brick	

Table 1: *List of gestures available to refine the part search.*

Users can also reduce the number of returned results during search, by selecting the desired type of part (Plates, Bricks, Tile, etc.), clicking on the small icons illustrated on the right of Figure 1.

4.3. Result Presentation

All existing applications for LEGO creation typically present the search results in an exhaustive text list. This way of presenting information is very uncomfortable and not very user friendly for users, since it forces them to recall rather than doing recognition. Users must preview several items in the text list before selecting one.

To overcome this, we use a suggestion list to present search results, allowing users to quickly recognize the part they want. We decided to limit the preview to 3 parts, mainly to deal with limited screen resolutions (800x600). A possible solution, in a near future, is to change the number of shown parts according to the size of the main window. On the right side of Figure 1 we can see a suggestion list, with the preview of three parts and the indication of the total number of parts in the list. On the right of this we have nine small buttons, which allow us to filter the results by category.

5. Constructing LEGO Models

Although, the addition of our retrieval approach makes the finding and re-use of LEGO parts easier and faster, we still need mechanisms to simplify the positioning, assembly and manipulation.

We observed users doing LEGO constructions using real parts to categorize the main constraints involved during the process. We identified three constraints: connections between parts, gravity and snap-to-grid. In our solution we relaxed the idea of connections and consider that two parts are connected, if and only if, one is above the other.

5.1. Assembly

Connections between parts are created when a part is placed over another. While users move a part or group of parts, our system detects collisions between them, using the bounding boxes. When a collision is detected, the system automatically places the moving part(s) on top of the static parts and connects them.

Internally, each part knows the parts that are above and below. This way, it is easy to propagate the movement to parts that are above the moved one. To improve the efficiency of the system, we associated a group number to parts that are connected. Thus, when doing collision detection, only parts from other groups are checked.

5.2. Transformations

Since our system has information about the connections between parts, we can perform transformations as we do in the real world. For instance, if we move a part, all the parts that are connected (on top) are also moved, due to the propagation of movement. First, only the moving part is processed. Then, it propagates the translation to the parts that are above. Next, the parts, which are above the moving part, propagate the translation to all their connected parts. To avoid circular dependencies and to guaranty that a connected part is moved only once in a movement, we use a flag to mark parts already processed.

Before the movement is made, and the translation calculated, collision detection is performed between the moved parts and other parts in the model. This has the advantage of detecting when a connected part, that is moving, but not selected, collides with other objects. When a collision is detected the system places the part that is moving on top of the collided part.

The rotation is done in a similar way as translation. The rotated part is rotated and the rotation is propagated to all connected parts. However, in this case no collision detection is performed.

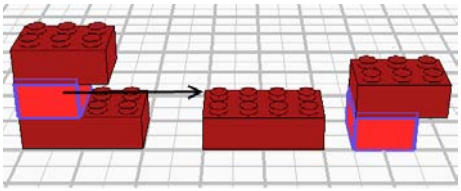

Figure 5: *Application of the gravity when moving parts.*

5.3. Snap-to-grid and Gravity

Due to the particular characteristics of LEGO parts, we implemented a snap-to-grid constraint that is applied when users place a part. The system automatically changes the part position to fit the grid in the 2D plane.

Additionally, we created a gravity mechanism that mimics the real interaction of parts. Thus, when users remove a part from a model, the parts that are above it fell over (see Figure 5). The algorithm works like this: While a part and all those that are connected to it are in the air, the algorithm moves them down one step. Then, it performs collision detection, to see if the parts are on top of other parts. If exists a collision, then parts that collide are fixed on top of the collided parts. If it does not collide, the cycle begins again, until there are no more parts in the air.

6. Calligraphic Interface

To connect the different components described above we developed a calligraphic interface to allow the creation of LEGO models using sketches, gesture commands and retrieval. We designed an application with a simple and easy to use user interface, with the most used commands available in toolbars (see Figure 1).

6.1. Sketch Recognition

The searching and manipulation of parts are done using calligraphic input. To search for a part users can use a continuous sketch or draw a set of incremental sketches defining lines. We call continuous sketch, for instance, when users draw two lines in the base grid with just one stroke, to specify a part (see Figure 3). In these cases the system recognizes the two lines, by identifying the points that are collinear and breaking the stroke in different lines when this condition changes.

Incremental drawing is used when users sketch lines one at a time. These lines are used mainly to specify the width and depth of a part. Additionally, users can also define the height of the part by drawing a vertical line, or refine the search by drawing specific gestures inside or outside the part, as explained in Table 1 and illustrated in Figure 4. These individual gestures are recognized using the CALI library [FPJ02].

Drawn sketches and gestures recognized by CALI are all in 2D. To convert them into 3D we use a common technique, called the unprojection of points, where 2D points are converted into 3D points by computing the intersection of a ray from the user point of view with a plane.

In our application we use a subset of the gestures recognized by CALI, such as, Triangles, Rectangles, Diamonds, Circles, Ellipses, Lines and Delete. These gestures have different meanings, depending of the state of the system. If the system is in the edition mode, all gestures are passed to the retrieval component and a search is performed or refined, as illustrated in Figure 4.

To select parts, users can perform the lasso command, by drawing a circle around an existing part, do a simple click or use the CTRL+click to select multiple parts.

To delete parts, users can perform a scratch over the parts they want to delete. The system identifies the delete gesture using CALI and then removes the affected part(s) from the model. When parts are deleted, the gravity constraint is activated and parts are moved to the correct position.

6.2. Parts Manipulation

To represent the parts graphically we used the representation provided by LeoCAD. However, we implemented a new manipulation mechanism, based on dragging and snapping, to fit the needs of the calligraphic input. To accomplish this we use the cursor position to unproject to the base grid. This way when moving parts the cursor is always mapped to the base grid, and the part to the cursor.

Due to the incremental drawing paradigm, the system accepts the edition of an inserted part in two different ways. One, changing the dimension of a part, by drawing a line starting in any bounding box corner. And second, by drawing a gesture over a selected part, as illustrated in Figure 4.

The base grid is used to simplify and to improve the positioning of parts. We use thin lines to represent one LEGO unit and thicker lines to represent two LEGO units. Additionally, the application changes the grid dimension according to the camera and parts position.

Our system also supports the copy operation, which can be applied when there are parts selected. To execute this command users need to press the CTRL key while moving parts. Feedback is given to users by changing the shape of the cursor to a "+" and the surrounding lines of parts, to convey information about selection.

The color palette of LSketchIt (see bottom of Figure 1) is composed by a small number of colors, since we found out that these are the most used colors (black, white, red, gray, yellow and blue). We also included the correspondent transparent colors for each opaque color.

6.3. Camera Manipulation

For camera manipulation, we keep most of the functionality from LeoCAD, changing only the camera pan, and the interface to access its controls, because it was very difficult to use. Our solution offers a mechanism to manipulate the camera similar to the dragging of parts. The camera position is mapped to the ground, when the user moves it, and the camera follows the movement incrementing its position and providing the concept of camera dragging.

There are two ways to rotate the camera in LSketchIt, using the widget's buttons in the camera toolbar or using the orbit function. The orbit function can be accessed by drawing a circle in an empty area of the window. After the orbit circle appear, users can rotate the camera in any direction using the pen.

7. Experimental Evaluation

To evaluate our system, we decided to compare it to the LeoCAD modeling tool. To that end, we performed tests with ten users, who had already played with physical LEGO parts, but had never used any software tool for creating LEGO constructions. Each user performed three tasks on both applications. However, to prevent users from learning tasks in one application and perform quicker on the other, five users performed the tasks first on the LeoCAD and the other 5 performed first on LSketchIt.

The model to create in the first task was very simple and was used mainly for users to get used to the applications. The second and third tasks had a higher complexity and were used to evaluate the assembly, searching and manipulation of parts, and camera operations.

Before executing the tests with users, we conducted a pilot test with two people to validate the experience protocol and to check if the predefined times for each task were reasonable.

The majority of the participants were males (80%), with age between 20 and 25 (70%) and with superior degrees (80% BSc and 10% MSc). From the ten users only two (20%) used a TabletPC or any other calligraphic device before this experiment.

In the next sections we present and analyze the results collected during the tests with users. We collected information about the time to complete tasks, time spent searching parts, errors and number of movements applied to parts. While users were executing the tasks we captured the screen for later analyzes.

7.1. Time to Complete Tasks

We measured the time each user took to complete each task on both applications. Achieved results are presented in Figure 6. As we can see, our system performs better than LeoCAD, being the difference more significant for Task 3, where users had to create the complex model presented in Figure 1.

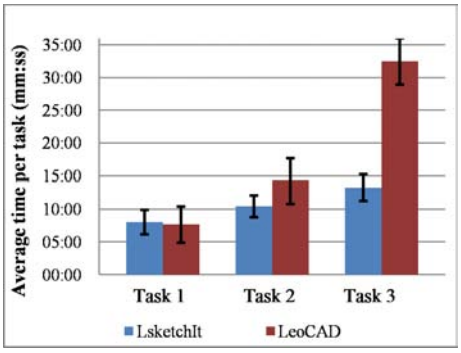

Figure 6: *Average time to complete each task.*

Comparing the times for the first and second task, we can see that there is no significant difference between the two tools. In task one LSketchIt average time is only 5% higher than in LeoCAD, while for task two the average time of LSketchIt is 27% smaller than in LeoCAD.

However, the result for the last task was particularly interesting, since the majority of the users did not finish it in the predefined time. In LSketchit 60% of the users finished the task in the expectable time, while the other 40% managed to complete more than 90% of the task. In LeoCAD, only 20% of the users finished the task, and only 40% completed more than 90% of the task. So, while in LSketchIt all users managed to conclude more than 90% of the task, in LeoCAD only 60% achieved this value.

The average time for Task 3 presented in Figure 6 was computed using just the times from users who finished the task. This small number of users caused a bigger standard deviation than previous tasks, but we can still compare both applications and see that users perform 41% faster using LSketchIt than using LeoCAD.

7.2. Searching Times

Currently, one of the main drawbacks of existing applications is the time users spent to locate a specific part to include into their constructions. We measured the time users spent searching for a part, to see if LSketchIt solves this problem.

Figure 7 presents the searching times for individual parts. Note that the chart is in a logarithmic scale of 2, to make the reading simpler. We show the minimum, the maximum and the average time users took for searching a part on each task. As we can observe, LSketchIt outperforms LeoCAD in all tasks and in all measures. It is also interesting to notice that the difference is bigger in the first task. This is explained by the simplicity of the parts used in the first task, that appear in the first set of parts of the suggestion list, without the need

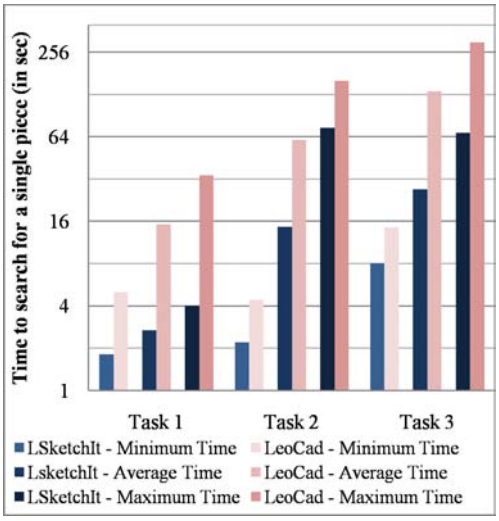

Figure 7: *Searching times on each task.*

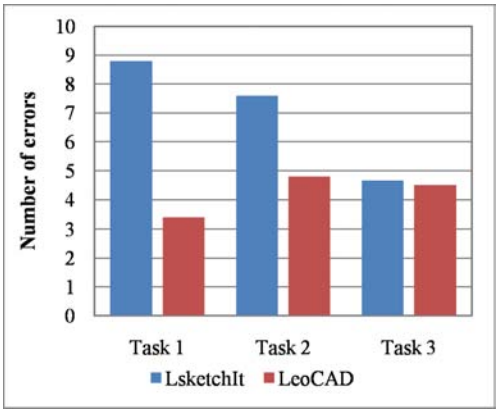

Figure 8: *Average number of errors per task for each application.*

to scroll. Thus, it was almost immediate to select a part in LSketchIt.

From Figure 7 we notice that the difference between the two applications is almost constant for the 3 tasks. So, the big difference registered in the time needed to complete Task 3 (see Figure 6) is due to manipulation. We can conclude that our snap-to-grid, gravity, collision detection and connection mechanisms, give some advantage to users when they have to create complex models, as the one from Task 3. Our system reduces the manipulation time, making the creation of complex models easier and faster.

7.3. Errors

During the execution of tasks we also count the number of errors that users did (see Figure 8). Contrary to our expectations, users committed to many errors with LSketchIt. In the first task users did 8.8 errors on average using LSketchIt, in opposition to 3.4 errors in LeoCAD, a difference of 159%. In the second task the number of errors reduced to 7.6, and in LeoCAD increased to 4.8, a difference of 58%. In the last task the number of errors reduced to 4.6 and in LeoCAD was registered an average of 4.5 errors, showing a small difference of only 2%.

To identify the cause for so many errors, we analyzed the screen captures. We notice that the main cause for the higher number of errors in LSketchIt was the difficulty that users felt in using a calligraphic device (70%). We believe that this is due to the lack of experience in using calligraphic devices (Tablet PCs and/or digitizing tablets). Another cause for errors was the difficulty in identifying the correct state of a part (20%). It was hard to distinguish between a part being edited or already created.

In LeoCAD, the main problems were due to its interface. Users had difficulty in finding the correct action, in locating the desired part and in placing parts in the correct place (sometimes parts were placed below the grid base).

We can observe from Figure 8 that the number of errors decreases from Task 1 to Task 3. Users learned how to use the calligraphic device, committing less errors. We believe that the number of errors in LSketchIt will largely reduce if users had have a training session to get used to the calligraphic device and interaction.

7.4. Coloring and Number of Movements

Finally, we count the number of movements applied to parts and the use of the coloring tool. The color function was less used in the LSketchIt tool than in LeoCAD (less 33% in Task 1, 41% in Task 2 and 26% in Task 3), to achieve the same result. The number of movements were also smaller in LSketchIt than in LeoCAD (less 42% in Task 1, 34% in Task 2 and 10% in Task3). With these results we can conclude, that to complete a task in LSketchIt, users perform less operations than with LeoCAD, making the interaction simpler and faster.

7.5. Satisfaction Evaluation

After performing the three tasks, we asked participants to evaluate both applications in terms of satisfaction, by filling a short questionnaire.

In general, users liked more the LSketchIt prototype than LeoCAD, giving the following grades. LSketchIt: 60% Good and 40% Very Good. LeoCAD: 60% Bad and 40% Good. When we asked about searching and insertion of parts, LSketchIt outperformed again the LeoCAD with 40% Very Good and 60% Good, against 80% Bad and 20% Good for

LeoCAD. The importance of connections and gravity between parts was also surprising, 60% find them Important and 40% found them Very Important. Finally, we asked if users would use the program in the future, and every users answered Yes for LSketchIt, and only 50% would use LeoCAD.

8. Conclusions and Future Work

We started this project with the objective of creating a simple and easy to use tool for LEGO constructions. After analyzing existing tools we found out that in most of them it was hard to construct LEGO models. Their user interfaces were difficult to use and do not mimic the real interaction users are used to have with real parts. Another problem of these tools was the searching of parts, which is a slow and painful process.

We also researched more general 3D modeling applications, to study their constrain and modeling mechanisms. After analyzing this state of the art, we concluded that a simple relationship between parts and a simple constraint solver using propagation of constraints, should suit users needs.

To solve the searching problem, we developed a retrieval technique that uses sketches to specify the characteristics of the part to locate. Search results are presented in a suggestion list, organized by categories. This in combination with a calligraphic interface to manipulate parts and the camera produced a simple and easy to use application.

To evaluate our prototype we compared it to the LeoCAD application. Results show that our system is able to reduce the construction time, of simple and complex LEGO models, and that users were very satisfied with the tool. Users considered specially important the concept of connected parts and gravity, because it mimics the real interaction.

Contrary to our expectations, we observed a large number of errors in LSketchIt, especially on the first tasks. The number of errors decreased for Task 2 and 3, which we think was due to the adaptation of users to the calligraphic device. We believe, and we want to test it, that if we performed the tests with users accustomed to calligraphic devices, the number of errors would be smaller.

Acknowledgements

This work was funded in part by the Portuguese Foundation for Science and Technology, project Augmented Decoration, POSC/EIA/59938/2004.

References

[BS06] BARADARAN H., STUERZLINGER W.: A comparison of real and virtual 3d construction tools with novice users. In *Proceedings of the International Conference on Computer Graphics & Virtual Reality, CGVR'06* (2006), pp. 10–15.

[CNJC03] CONTERO M., NAYA F., JORGE J., CONESA J.: Cigro: A minimal instruction set calligraphic interface for ketch-based modeling. In *Computational Science and Its Applications - ICCSA'03* (2003), vol. 2669 of *Lecture Notes in Computer Science*, Springer, pp. 549–558.

[FPJ02] FONSECA M. J., PIMENTEL C., JORGE J. A.: CALI: An Online Scribble Recognizer for Calligraphic Interfaces. In *Proceedings of the 2002 AAAI Spring Symposium - Sketch Understanding* (Palo Alto, USA, Mar. 2002), pp. 51–58.

[IH01] IGARASHI T., HUGHES J. F.: A suggestive interface for 3d drawing. In *UIST '01: ACM symposium on User interface software and technology* (New York, NY, USA, 2001), ACM, pp. 173–181.

[IMKT97] IGARASHI T., MATSUOKA S., KAWACHIYA S., TANAKA H.: Interactive beautification: A technique for rapid geometric design. In *UIST'97: ACM Symposium on User Interface Software and Technology* (1997), pp. 105–114.

[Lac] LACHMANN M.: Mike's lego cad. Website. http://www.lm-software.com/mlcad/ (Accessed September 2007).

[ldd] Lego digital designer. Website. http://ldd.lego.com/ (Accessed September 2007).

[leo] Leocad. Website. http://www.leocad.org/ (Accessed September 2007).

[PBJ*04] PEREIRA J. P., BRANCO V. A., JORGE J. A., SILVA N. F., CARDOSO T. D., FERREIRA F. N.: Cascading Recognizers for Ambiguous Calligraphic Interaction. In *Eurographics workshop on Sketch-Based Interfaces and Modeling* (Grenoble, France, Aug. 2004), Eurographics Association, pp. 63–72.

[SI07] SHIN H., IGARASHI T.: Magic canvas: interactive design of a 3-d scene prototype from freehand sketches. In *Proceedings of Graphics Interface, GI'07* (2007), ACM, pp. 63–70.

[ske] Google sketchup. Website. http://sketchup.google.com/ (Accessed September 2007).

[Sut63] SUTHERLAND I. E.: Sketchpad:a man-machine graphical communication system. In *AFIPS Spring Joint Computer Conference* (1963), pp. 329–346.

[ZHH96] ZELEZNIK R. C., HERNDON K. P., HUGHES J. F.: Sketch: an interface for sketching 3d scenes. In *SIGGRAPH '96: Conference on Computer graphics and interactive techniques* (New York, NY, USA, 1996), ACM Press, pp. 163–170.

[ZL07] ZOU H. L., LEE Y. T.: Constraint-based beautification and dimensioning of 3d polyhedral models reconstructed from 2d sketches. *Computer Aided Design 39*, 11 (2007), 1025–1036.