CEIG - Spanish Computer Graphics Conference (2018)
I. García-Fernández and C. Ureña (Editors)

LeoMCAD: A Lego-based Mechanical CAD system

Francisco González García, Jesús Amador Pérez Martín, and Gustavo Patow

ViRVIG, Universitat de Girona, Spain

Abstract

Mechanical Design (MCAD) tools are used for creating 3D digital prototypes used in the design, visualization, and simulation of products. In this paper we present LeoMCAD, a Lego-based mechanical system designed to be used as an education tool both for kids and Lego hobbyists; but which features a novel solver that naturally and seamlessly computes the interaction between the pieces that build-up a given model, solving an otherwise complex forward kinematic system of equations in a much simpler way. The results show how our system is able to cope with situations that would produce dead-lock situations in more advanced commercial systems.

Keywords: Mechanical CAD, Procedural Modeling, Solver

1. Introduction

The potential to use specialized software for the analysis and design of mechanisms has been recognized since the early days of computers [Fre54]. Some modeling systems, for instance, implement function-oriented and shape-oriented approaches, where objects, assemblies and positional relationships are represented as nodes in a graph structure, and an explicit or procedural representation is used for shape primitives [WLPL*80, GM94]. Despite the significant benefits presented by such tools, creating complex mechanisms, such as those needed to animate mechanical characters, currently requires expert designers. In this paper we present a simple, easy to use and flexible mechanical CAD (MCAD), based on the popular Lego bricks, bringing all the possibilities of such systems to non-expert, all-age users. However, given the generality of our solution, the method we present here can be used at any level, being feasible to be easily incorporated into more professional tools.

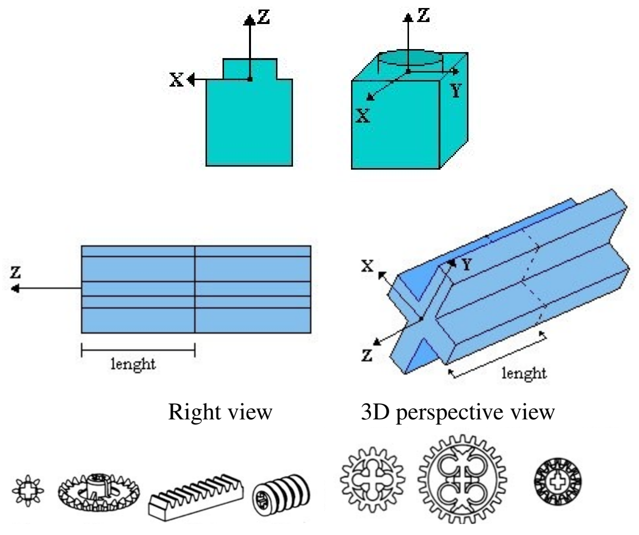

Figure 1: *Parts and connector types. Top row: point-like (studs). Middle row: Segment-like. Bottom row: gear connectors (regular, crown, rack, worm, and two different gears and a bevel gear).*

2. Previous Work

Mechanical Animation. Although most 3D CADs do not include information about how their parts move and interact, a few commercial packages help users create mechanical animations in order to evaluate functional aspects of an assembly design (e.g., Autodesk Inventor [Aut18], SolidWorks [Das18], Solid Edge [Sie18]). However, most of these tools still require the user to manually specify information for all (or many) of the assembly parts, including motion parameters and interaction constraints between the parts. In contrast, our approach infers such information directly from geometry with far less user assistance.

Motion Analysis. There exists a variety of methods for analyzing the geometry of mechanisms in order to extract the kinematic constraints that define their motions. Mitra et al. [MYY*10] presented an automatic approach for the visualization of the operation of complex mechanical assemblies. Reverse engineering of scanned devices has been also a source of interesting motion analysis approaches [DVVR07]. Bacher et al. [BBJP12] analyze the creation of a printable articulated model from input geometry by analyzing a skinned mesh. Cali et al. [CCA*12] presented an intuitive user interface to control the placement and range of motion of joints.

DOI: 10.2312/ceig.20181163

While the resulting characters can be posed, these methods do not address the challenge of animating or "playing" with them.

Constraint-based design. Krecklau and Kobelt [KK11] developed a procedural modeling tool for interconnected structures like bridges, roller coasters and catenaries, computing the angles between of the elements in a rigid chain to achieve a given required shape. Zhu et al. [ZXS*12] introduced a new technique to generate mechanical toys solely based on the motion of their features. Ceylan et al. [CLM*13] proposed an automatic algorithm to transform a motion sequence of a humanoid character into a design for a mechanical figure that approximates the input motion when driven with a single input crank. Similarly, Coros et al. [CTN*13] presented a computational design method of the generation of mechanical characters, based on the transformation of a user-provided animation of the model into an optimized mechanism that closely follows the prescribed movements. Megaro et al. [MTN*15] presented an optimization-based solution that generates stable motions for legged robots of arbitrary designs. Later on, Megaro et al. [MKS*17] presented an optimization-based approach for the design of cable-driven kinematic chains and trees.

The main difference with the previous works is that our system is a general-purpose, simple-to-use, constraint-based mechanical CAD system for Lego parts, where the kinematics of the interactions are computed thanks to a simplified solver.

3. Connectors

In our system, the basic construction unit is a (Lego) part (or piece), as shown in Figure 1. Each piece has connectors, which are the elements within each piece that enables assembling between pieces: a connector is a part of a piece that can assemble with the connectors from other pieces. For example, the 2×2 brick (3003) connects through its Stud and Stud Inlets. See Figure 1. The goal of our system is to model the connections and interactions between pieces by modelling the individual connectors, their interactions and their behaviours, thus naturally solving a complex kinematical system without ever resorting to a full-system solution.

Types of connectors The connectors can be classified in two main categories: *Point-like* connectors can be interpreted like a point in a 3D space, as illustrated in Figure 1 for the case of a Stud. Studs and Stud Inlets belong to this family. These are rigid connections with 0 degrees of freedom. *Segment-like* connectors own a body of a certain length l that allow them to assemble to N opposite connectors at the same time (see Figure 1, bottom). Example of this type are Axles, Rods and Cylindrical Holes. Special note must be taken to the fact that Axles actually are two connectors, one from each extreme up to the middle. Once connected, they usually admit one degree of freedom as longitudinal translations along their main axis. Finally, *Gears* have an outer circumference where other gears can be connected. In general, gears can rotate around their main axis, which depends on the specific gear being considered.

Connectors can also be classified as simple or multiple: the former can only be connected with one and only one connector, while the latter can be connected to several connectors at the same time. All the available connectors in the current version are (See Figure 1):

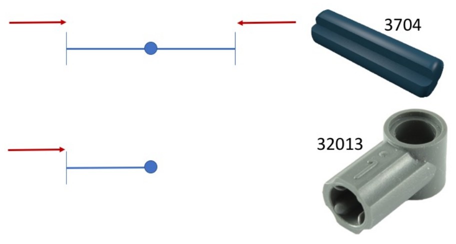

Figure 2: *Schematic representation of two different axles (3704 and 32013). Red arrows are entry points for new connections. For 3704, the blue dot shows the union of the two brother connectors.*

- **Stud**. The Stud connector is the male of the Stud Inlet and Cylindrical Hole connectors. The kind of assembly that a Stud can make is simple: a Stud connector can only be joined with only one Stud Inlet or one Cylindrical Hole at the same time.
- **Stud Inlet**. The kind of assembly that Stud Inlet can make is the same as the Stud connector: simple connection with a Stud.
- **Axle**. It is the male of the Axle Hole and Cylindrical Hole connectors and has multiple (many Axle Hole or Cylindrical Hole connectors can be connected to an Axle connector at the same time) or simple assembling.
- **Axle Hole**. As the Axle connector, the Axle Hole can assemble on a multiple or simple basis.
- **Cylindrical Hole**. This connector is another female to the Axle connector and can also make multiple or simple assembling. Connections with Studs at the extremes of the Cylindrical Hole are also possible.
- **Gear**. Gears can be of several types, being the most common the **regular** one, the **crown**, the **rack**, the **bevel**, and the **worm**.
- **Other Connectors**. Although other connectors, such as **hinge** connectors, have not been implemented, it is simple to see that they can easily be incorporated in our system.

Connector brotherhood For convenience, certain segment-like pieces, such as axles and holes, must be subdivided in two brother connectors. Two connectors will be brothers if they belong to the same piece, they are segment-like connectors, they are coaxial on the same axis, and they are consecutive. See Figure 2.

Connector state Connectors can be classified as either free or busy, being selectable only the free ones. Busy connectors will have a reference to the connector (or set of connectors for segment-like connectors), with which they are assembled. See Figure 3-left. A connector state can be classified by the type of the connectors (e.g., a point-like connector will be considered free if it has no connector connected, and busy otherwise.) Segment-like connectors, as can have multiple connectors connected to them, can be considered as a free and busy at the same time.

4. Assembling

Assembling is the process of connecting two pieces by its connectors. First, the user must select the connectors to assemble. We can select a connector by "clicking" on it with the left mouse button.

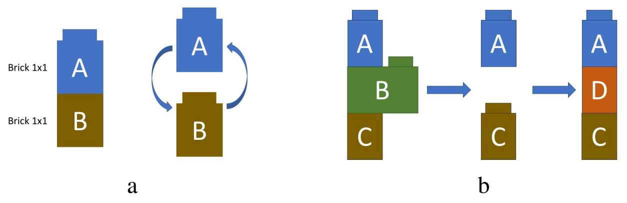

Figure 3: *Connection creation. a) The Stud Inlet from piece A knows that it is assembled with the Stud from B and the other way round. b) Assembling propagation. After the initial model (left) was created, B is removed (middle), so now the Stud from C, and the Stud Inlet from A are free. Finally, when adding D, the Stud from D must be connected with the Stud Inlet from A, and the same for the Stud from C and the Stud Inlet from D. Once the user creates one of the two connections, the other one will be created automatically.*

We want to clarify that Axle, Axle Hole and Cylindrical Hole connectors can only be selected by their extremes, where they can perform connections. Similarly, a Gear can only be connected trough its outer circumference. For the assembling, the user selects a connector by double clicking on it. This will be the *static* connector and the other the *dynamic* on, as explained below. After verifying the compatibility of the two connectors, the system applies a series of constraints:

- The axis of the *dynamic* connector is aligned to the *static* one.
- Their respective origins are matched by translating the dynamic connector.
- For connectors that require axis alignment, such as Axles and Axle Holes, a rotation around their longitudinal axis is applied to align both coordinate systems.
- For gears, perpendicularity (for racks and worms) or co-planarity (for the other gears) are also enforced. Also, their teeth are aligned, mostly for aesthetic purposes.
- Finally, the status of the connectors are updated, as well as the lists of empty and busy connectors at each piece.

Once finished, the system automatically propagates the assembling to all the candidate connections among all intervening pieces, which consists on recursively verifying if there are more connections resulting from the first one, see Figure 3-right. This is done with a typical graph depth-first search algorithm, that traverses every connecting piece and checking if its connectors satisfy the corresponding constraints, which are not enforced this time. To prevent robustness issues during propagation, our system takes advantage of the sizes involved in Lego pieces, and verify connections using a safety range ε: if two connectors are closer than ε, and if a valid connection is available, then the two connectors are assembled.

5. Solver

One important aspect of any MCAD implementation is its solver. Ours is based on an implicit connector-based formulation. One simplifying assumption of our solver is that the constraints need to be enforced only at the assembling stage, and not at the simulation stage. Once two connectors are assembled, their alignments are not verified again, which strongly simplifies any further kinematic computation.

Once users interact with an assembled model, they will do it by rotating or translating a piece, which immediately informs every connector of the transformation. Then, recursively, all the connectors will inform the opposing ones that there has been a transformation, which in turn will be passed to the owner pieces, and then recursively to the whole assembly. To prevent infinite recursions, again a graph-based depth-first strategy was used to guarantee no transformation was applied more than once to any piece. It is important to remark that, when possible, sliding is taken into account using only the longitudinal part of the translation. If a collision is detected, then the pieces start to move together with the full transformation, as described below.

Translating and rotating assembled pieces. As mentioned, translations and rotations on one piece are recursively transferred over the whole assembled model. However, it is important to mention that, if you rotate an Axle, connected to a Cylindrical Hole, and the rotation is over the axis of the hole, it will actually *not* propagate the rotation because the Axle connector can freely rotate over the axis of the Cylindrical Hole, and the other way round, constituting what is usually called a *revolute joint* in more traditional systems. See Figure 4, left. Also, as an approximation, our system does not detect collisions during rotations.

Sliding Axle, Axle Hole or Cylindrical Hole connectors. When a segment-like connector is assembled to another connector of the same type, we can slide one of the two connectors over the other. This is usually called a *Prismatic Joint* in traditional CAD systems. This is a very useful option because we can only make assembling using Axle, Axle Hole and Cylindrical Hole connectors by their extremes. So when they are assembled, we can slide and select in which position we want them, as shown in Figure4 right.

Notice that, when a connector slides over another, it can result in the formation of new connections and the elimination of others, and also the possible changing of the connector state. See Figure 4. Thus, in every slide action we must verify all the assemblies between the sliding connector (the dynamic connector) and the rest of the model (the static connectors).

Another important thing to comment about the sliding process is that, when we slide a connector and its piece has some others connectors assembled to it, we must translate recursively the connected pieces to the one that has the sliding connector. See Figure 4.

Collision detection. We perform this verification only by the translation and sliding processes, and in particular only for the Axle-Cylindrical Hole pair when one of the two connectors is translating along its main axis. Rotations do not undergo these verifications. At Figure 5 we can see a complex example of how the a piece with a Cylindrical Hole (CH_2) is translated to the left. The piece that owns the CH_1 and CH_2 (piece P_2) is connected indirectly to the piece that owns the Cylindrical Holes CH_3 and CH_4 (Piece P_3). The collision detection can resolve cases like this, so when the CH_3 collides with the Axle Hole AH_2, the whole model will translate because the Cylindrical Hole "pushed" the others because of the translation of CH_2.

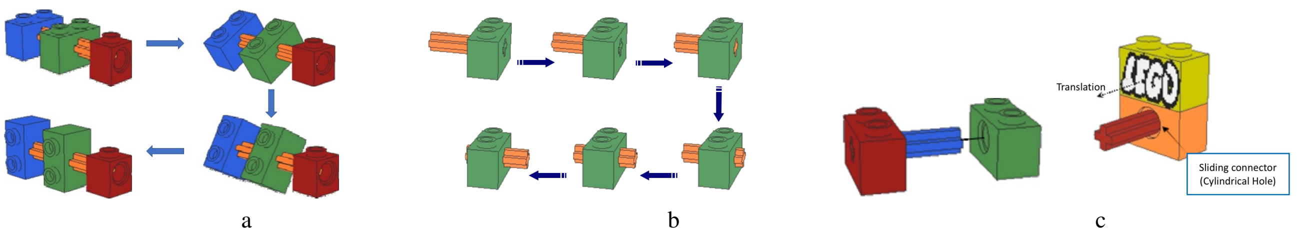

Figure 4: *Movement Examples. a) An Axle freely rotating over a Cylindrical Hole. b) Example of the sliding process. c) When we slide the blue Axle to the right, the assembling with the red Axle Hole disappears, while a new connection between the Axle and the green Axle Hole appears. However, when we slide the Cylindrical Hole over the Axle, the piece connected to the former must also be translated.*

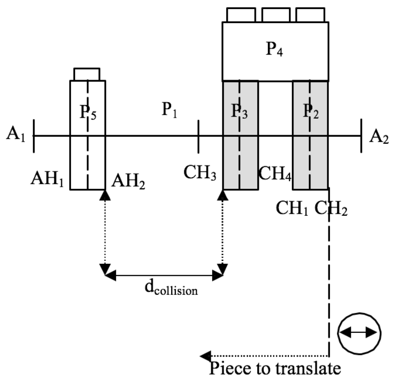

Figure 5: *Collision Example. Piece Diagrams: P1: 3704, P2: 6541, P3: 6541, P4: 3622, P5: 32064.*

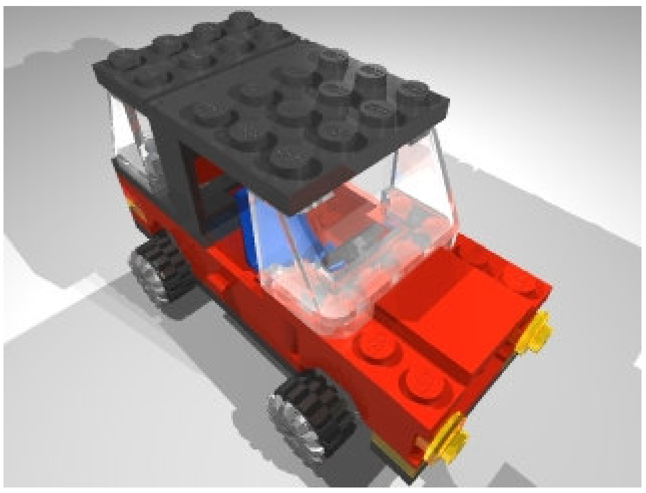

Figure 6: *A full model created with LeoMCAD.*

6. Results

All the images in this paper have been generated with our system. Figure 4-left, shows examples of transformations being applied to an object and correctly propagated to the whole model, taking into account the characteristics of each connector intervening in the propagated movement. Figure 4-middle and -right illustrate the sliding process, and how sliding can be used to generate new connections, or to disconnect two connected pieces. Figure 6 shows a complex example generated with our tool.

With our system we also performed a simple test case where more sophisticated tools, such as Autodesk's Inventor [Aut18], cannot solve: the system is locked and the mechanism, however simple as it is, cannot move. See Figure 7. The reason is that the constraints used to generate the assembly are still enforced by the solver during the simulation, while our solver only uses the rotation relations and does not verify the set-up constraints after the initial assembly, thus avoiding lockout situations. Of course, it is possible to delete the constraints once the initial assembly is done, but then simple operations such as rotating the mechanism might result in a separation of the pieces, which does not happen in our solver.

7. Conclusions and Future Work

We have presented LeoMCAD, a simple, easy to use and flexible mechanical CAD, based on the popular Lego parts, bringing all the possibilities of such systems to non-expert, all age users. The system is based on a constraint-based assembling system, and interactions between the pieces are allowed thanks to a simplified solver that works at the connector level, never building the implicit full system of kinematic equations. The solver and the whole system work interactively, as the assembly propagation and the kinematic interactions are solved in a very direct way. Given the generality of our solution, the method we present here can be used at any level, being feasible to be easily incorporated into more professional tools. Possible lines for future work include the application of the presented technique to other, professional-level tools such as Autodesk Inventor [Aut18]. Another avenue for study is the generalization of the definition of assembly, allowing the use of interchangeable sub-assemblies which can be procedurally combined to generate more complex mechanisms. Finally, the use of inverse techniques as the one seen in urban procedural modeling seems a promising way to add new possibilities to the burgeoning area of Constraint-based design.

Acknowledgements

We would like to thank our reviewers for their insightful comments on the paper, as these comments led us to an improvement of the work. This work

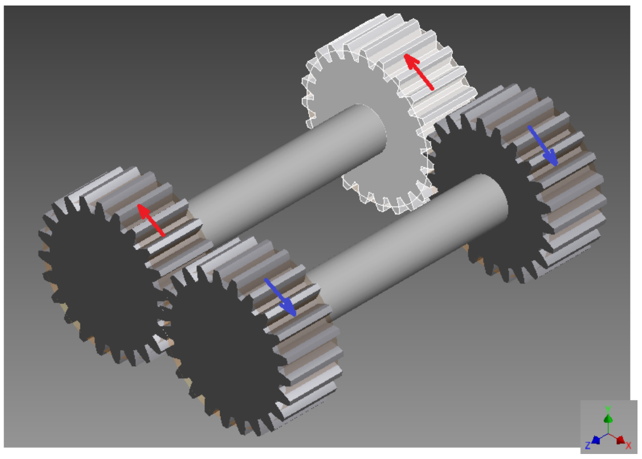

Figure 7: *A simple mechanism that our solver can simulate, but Autodesk's Inventor [Aut18] cannot.*

was partially funded by the project TIN2017-88515-C2-2-R from Ministerio de Economía y Competitividad, Spain.

References

[Aut18] AUTODESK: Inventor. `http://autodesk.com/inventor`, 2018.

[BBJP12] BÄCHER M., BICKEL B., JAMES D. L., PFISTER H.: Fabricating articulated characters from skinned meshes. *ACM Trans. Graph. 31*, 4 (July 2012), 47:1–47:9.

[CCA*12] CALÌ J., CALIAN D. A., AMATI C., KLEINBERGER R., STEED A., KAUTZ J., WEYRICH T.: 3d-printing of non-assembly, articulated models. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 130:1–130:8.

[CLM*13] CEYLAN D., LI W., MITRA N. J., AGRAWALA M., PAULY M.: Designing and fabricating mechanical automata from mocap sequences. *ACM Trans. Graph. 32*, 6 (Nov. 2013), 186:1–186:11.

[CTN*13] COROS S., THOMASZEWSKI B., NORIS G., SUEDA S., FORBERG M., SUMNER R. W., MATUSIK W., BICKEL B.: Computational design of mechanical characters. *ACM Trans. Graph. 32*, 4 (July 2013), 83:1–83:12.

[Das18] DASSAULT SYSTEMES: SoludWorks. `https://www.solidworks.com/`, 2018.

[DVVR07] DEMARSIN K., VANDERSTRAETEN D., VOLODINE T., ROOSE D.: Detection of closed sharp edges in point clouds using normal estimation and graph theory. *Comput. Aided Des. 39*, 4 (Apr. 2007), 276–283.

[Fre54] FREUDENSTEIN F.: *Design of Four-link Mechanisms*. University Microfilms, 1954.

[GM94] GUI J.-K., MÄNTYLÄ M.: Functional understanding of assembly modelling. *Computer-Aided Design 26*, 6 (1994), 435 – 451.

[KK11] KRECKLAU L., KOBBELT L.: Procedural Modeling of Interconnected Structures. *Computer Graphics Forum* (2011).

[MKS*17] MEGARO V., KNOOP E., SPIELBERG A., LEVIN D. I. W., MATUSIK W., GROSS M., THOMASZEWSKI B., BÄCHER M.: Designing cable-driven actuation networks for kinematic chains and trees. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (New York, NY, USA, 2017), SCA '17, ACM, pp. 15:1–15:10.

[MTN*15] MEGARO V., THOMASZEWSKI B., NITTI M., HILLIGES O., GROSS M., COROS S.: Interactive design of 3d-printable robotic creatures. *ACM Trans. Graph. 34*, 6 (Oct. 2015), 216:1–216:9.

[MYY*10] MITRA N. J., YANG Y.-L., YAN D.-M., LI W., AGRAWALA M.: Illustrating how mechanical assemblies work. *ACM Trans. Graph. 29*, 4 (July 2010), 58:1–58:12.

[Sie18] SIEMENS: Solid Edge. `https://www.plm.automation.siemens.com/en/products/solid-edge/`, 2018.

[WLPL*80] WESLEY M. A., LOZANO-PEREZ T., LIEBERMAN L. I., LAVIN M. A., GROSSMAN D. D.: A geometric modeling system for automated mechanical assembly. *IBM Journal of Rsearch and Development 24*, 1 (1980), 64–74.

[ZXS*12] ZHU L., XU W., SNYDER J., LIU Y., WANG G., GUO B.: Motion-guided mechanical toy modeling. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 127:1–127:10.