Generating Stable, Building Block Structures from Sketches

Matthew Stephenson[1], Jochen Renz[1], Xiaoyu Ge[1], and Peng Zhang[1]

Australian National University, Research School of Computer Science, Canberra, A.C.T. 0200, Australia, matthew.stephenson@anu.edu.au

Abstract. This paper presents a structure generation algorithm which converts rough human drawings into stable structures comprised of rectangular blocks, suitable for physics-based 2D environments. Generating viable structures for a physics-based environment imposes many additional requirements above those of most traditional sketch-based domains. Our method is sophisticated enough to deal with these requirements, while still ensuring that the generated structure accurately represents the original sketch. We describe and implement a framework for this process, allowing inexperienced users to create complex structures with ease. Multiple structure possibilities are identified for a single drawing and are then compared based on their similarity to the original sketch using a heuristic value. We evaluate our approach by investigating its ability to replicate structures for the video game Angry Birds, based on human drawn sketches of the original levels.

1 Introduction

AI assisted generation of digital content with minimal or reduced human input, also known as procedural content generation, has become an increasingly popular area of research over the past few years [35]. This process allows for the fast and efficient generation of suitable content, without the need for experienced designers or developers. However, the methods implemented for achieving this typically have a very limited or unintuitive range of options for designer control [17, 33]. This makes it difficult for the average user to effectively design and create their own content. One possible solution to this problem is to implement a sketch recognition and understanding system, allowing users to interact with programs through the use of hand drawn inputs [7]. The use of a sketch-based interface for the automatic generation of content, referred to in this paper as sketch-based generation, allows users to design and create their own content quickly and intuitively.

Previous applications and research around sketch recognition has been conducted in many different areas, including designing analog electrical circuits [1], creating UML diagrams [16], drawing chemical molecule structures [27], and solving physics-based problems [12]. Several different algorithms have also been proposed to generate virtual content for video games from 2D human sketches. These include designing maps for strategy games [22], levels for 2D platformers [32], building 3D game worlds [31], modelling human characters [21], and creating virtual garments [37]. However, none of these methods have had to consider whether or not the result is feasible within a realistic

physics-based environment. This type of environment places additional restrictions on the generated content, such as a limited number of resource options or requiring that the result be stable. Several previous programs for creating content within a physics-simulation have been developed, such as CogSketch [13], SketchyDynamics [5] and PhysicsBook [4], but these also make no attempt to correct or fix the generated content and simply recreate exactly what the user has drawn.

In this paper we develop an approach to generate stable and feasible structures for a 2D physics environment, based solely on human sketches. These input sketches comprise of multiple axis-aligned, rectilinear (aka. orthogonal) polygons, that can be placed next to and on top of each other. The output structures generated based on these sketches are created using rectangular building blocks, with a pre-set number of different block dimensions (shapes) available. Each generated structure should satisfy the requirements of the environment (stable on flat ground, no overlapping blocks, etc.) while representing the original sketch's design as closely as possible. Unlike prior sketch-based interfaces for creating content in physics-simulations, our proposed generation process does not merely replicate the design of the input sketches, but also ensures the physical viability and constraints of the generated structure are maintained. We believe that this task is sufficiently complex and novel to be worthy of investigation, posing many different challenges for the areas of physical and spatial reasoning.

1.1 Angry Birds

The specific example we will use to demonstrate the benefit of solving this problem is for the popular video game Angry Birds. This game utilises a 2D physics-based environment and its levels often consist of one or more structures composed of multiple rectangular blocks, providing a perfect example domain to evaluate our approach. Angry Birds has also been used for multiple AI competitions focused around generating and solving levels [28]. Multiple level generators for Angry Birds currently exist, the latest of which offer several options for designer influence and requirements [34, 11]. However, the level of control that designers have over the generated content is still very minimal, offering little more than some simple specifications such as the size of the structures or the number of block shapes available.

Another recent level generation paper for Angry Birds proposed a mixed-initiative generation system that allows the user to design structures using a built-in drawing tool [3]. This system is exceptionally primitive in its current form though, allowing users to only draw blocks using a predefined grid and requiring that all blocks have either a width or height of exactly one grid unit. This process essentially corresponds to users selecting which squares of the gird they want filled using straight lines of a fixed width, rather than sketching the whole structure's design in the traditional sense. This results in structures that are hugely simplified compared to more traditional Angry Birds levels. This method also offers no real analysis on the stability of the generated structures, leaving most of this to the human designer. Overall this approach can only be loosely called a sketch-based generation method, and can only create vastly simplified versions of structures that are atypical for Angry Birds.

The sketch-based generation system proposed in this paper allows for much greater designer control in terms of the look and overall aesthetic of the desired structures,

whilst still ensuring that the generated levels are feasible within the game's physics engine. We demonstrate that our generation method provides a fast and effective way of developing level prototypes, and that even inexperienced users can create detailed and personalised structures with ease.

2 Structure Generation Approach

In order to generate stable, building block structures based on human drawn sketches, several different sub-problems must be solved. Each of these can be treated as a separate task with multiple possible approaches and solutions. This section provides detailed descriptions and possible solutions for each of these problems, as well as other additional features that either improve the end result or reduce the generator's runtime.

Process Overview We first provide an overview of the entire generation process from original sketch to final generated structure. 1) We identify separate polygons within the sketch, as well as their corners. Based on this information, any non-rectangular polygons are split into multiple separate rectangular components. All these rectangles are then combined to make a full structure. 2) This structure is tested for stability, and suitably adjusted if need be. 3) The rectangles within the structure are grouped based on their position, size and shape, which helps improve the generation process. 4) Composite block shapes are created by combining multiple regular blocks together. 5) All rectangles are scaled to be closer in size to the actual block shapes available. 6) The final generated structure is recursively built one block at a time, by selecting for each scaled rectangle the block shape that is closest to its size and aspect ratio. If when selecting a block shape any of several requirements are violated (structure is unstable, blocks overlap, etc.) then the block is either moved or swapped out for a different block shape. This continues until a structure that satisfies all requirements has been generated. 7) The generated structure is evaluated using a similarity heuristic calculation between itself and the original sketch. Multiple different structures can often be created for the same sketch by changing certain generation parameters (e.g. scaling method, structural requirements, block adjustment options), which can then be ranked based on their similarity heuristic values.

2.1 Polygon Splitting under Stability

Problem: *Split a collection of sketched, roughly axis-aligned, rectilinear polygons into a collection of rectangles that mimics the shape of the original input sketch; with an optimisation criteria that the structure created by these output rectangles be stable on a flat horizontal plane under the influence of gravity.*

The first problem that we must solve is that of splitting polygons within our input sketch into rectangles. To extract the properties of each polygon from the sketch, we take advantage of the fact that any collection of non-intersecting rectilinear polygons can be uniquely determined based on its vertices [25]. It is therefore possible to recreate the shape of each polygon by simply identifying corners within the input sketch. For our program we found that the Shi–Tomasi corner detection algorithm worked well enough

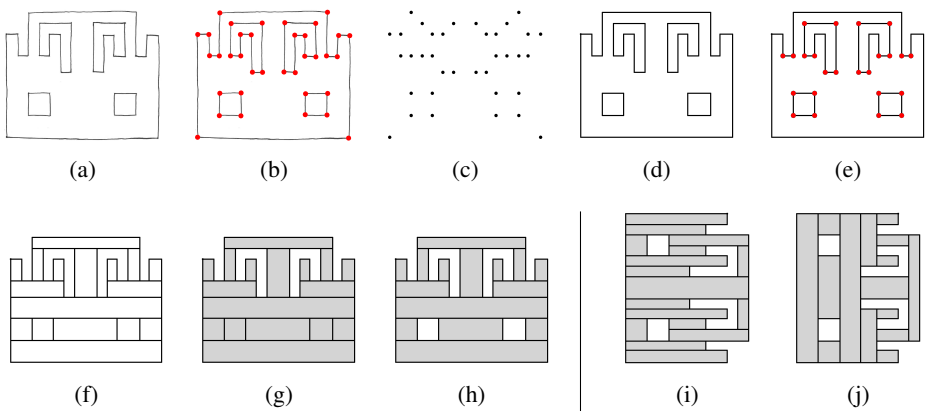


Fig. 1: Polygon splitting under stability: (a) an example polygon sketch, (b) corners detected using our chosen algorithm (red dots), (c) corners with similar x-axis or y-axis location values are made the same, (d) horizontal and vertical edges identified between detected corners, (e) ray casting used to identify concave corners (red dots), (f) horizontal lines added at concave corners, (g) rectangles that can be formed using these lines / edges are identified, (h) ray casting used to identify and remove any rectangles that are actually holes. Figure (i) shows the result of PSSA on a rotated polygon sketch, whilst (j) shows the negative result of using the same split lines for both the non-rotated and rotated input sketch (approach used by prior methods).

[29], but other more sophisticated methods are available [38, 6, 30, 39]. Note that if all polygons within the sketch are already rectangular, then a simple minimum bounding rectangle (MBR) detection algorithm is sufficient to identify the properties of each. We now attempt to replicate the shape of each identified polygon using only rectangles.

Several papers have already proposed solutions to this problem of partitioning rectilinear polygons [8, 26, 2, 15, 19, 10], but these methods have different optimisation criteria (minimum number of rectangles, polynomial time approximation, maximum smallest rectangle dimension, minimum stabbing number, etc.) and do not take the physical nature of our scenario into account. We therefore propose a new algorithm for polygon splitting under stability (PSSA). Accompanying diagrams for each step of PSSA are shown in Figure 1.

Polygon Splitting under Stability Algorithm (PSSA)

(a) Take as input a collection of roughly axis-aligned, rectilinear polygons, orientated such that the vertical axis is aligned with the gravitational force.
(b) Identify all corner positions P within the input using a chosen corner detection algorithm (e.g. Shi–Tomasi).
(c)
- For every point P_i in P, create an associated set S_i of all points in P that have x-axis location values within a certain number of pixels n of P_i's x-axis location.

  - If any two sets S_i and S_j share a common point ($S_i \cap S_j \neq \emptyset$), merge them together to make a new set S_{ij} that is associated with both i and j ($S_{ij} \leftarrow S_i \cup S_j$).
  - For every point P_i in P, make the x-axis location value for P_i equal to the average x-axis location value of all points in its associated set.
  - Repeat the above three steps but this time for y-axis location values.

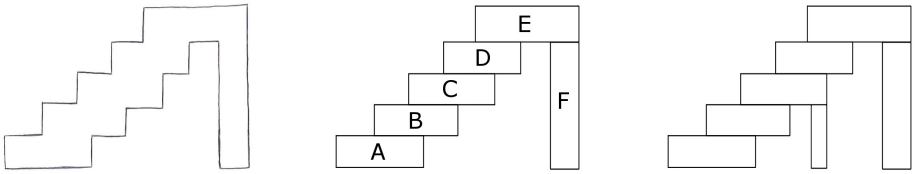
  This step is done to account for any slight imperfections in the sketch, allowing corners that were intended by the drawer to be the same in terms of their x-axis or y-axis location to actually be so (i.e. turning polygons that are roughly axis-aligned and rectilinear into polygons that are exactly axis-aligned and rectilinear).
(d) From these adjusted corner positions P we identify all horizontal and vertical edges E that connect them, using the method described in [25].
(e) Ray casting is used to identify which corners in P are concave (vertex points with an interior angle of 270 degrees) based on the number of edges in E that the ray intersects.
(f) Additional horizontal lines are added to E at each concave corner in P. These additional horizontal lines originate from each concave corner in both the left and right directions, stopping once they intersect another edge in E.
(g) Based on this collection of lines E we can create a collection of possible rectangles R that they can form.
(h) Ray casting is used to determine which rectangles in R are solid regions and which are holes, removing those that are the latter from R.

By following the steps outlined in PSSA we can divide up a sketch of one or more axis-aligned, rectilinear polygons into a collection of solid rectangular regions (R) that accurately represents its shape. Due to the fact that only horizontal lines are added to the sketch in step (f), we can guarantee that every rectangle created by PSSA touches another rectangle on at least one of its horizontal edges. This guarantee heavily increases the likelihood of R being stable, as it reduces the risk of certain rectangles having none or minimal support. This also means that the same polygon may be split differently based on its orientation, which is not the case for other prior methods. Figure 1 (i) represents the result of performing PSSA on the same polygon sketch from Figure 1 (a) but rotated 90 degrees. Even though Figure 1 (i) contains more rectangles than if we used the same split lines from the original non-rotated sketch, see Figure 1 (j), the result is far more stable (all rectangles supported from below). This demonstrates how important it is for an input sketch to be split differently based on its orientation, which is something that other splitting methods do not do.

In all subsequent sections of this paper, the term *block* will be used to refer to a solid rectangular region, and a collection of one or more axis-aligned, rectangular regions will be referred to as a *structure*.

2.2 Stability Analysis / Adjustment

Problem: *Estimate the stability of a structure that is resting on a flat horizontal plane under the influence of gravity, and if the structure is unstable propose a modification that makes it stable.*



(a) (b) (c)

Fig. 2: Original sketch (a), sketch after polygon splitting (b), and the adjusted sketch after stability analysis (c).

Once all the rectangles (blocks) from our input sketch have been confirmed, we next test for structural stability. Determining local stability for each block can be calculated quickly based on qualitative stability relations from the extended rectangle algebra (ERA) [40], but using this alone often results in many unstable structures being falsely classified as stable and vice versa. The actual stability of a given structure can be calculated exactly, but only if all the relevant physics parameters of the involved objects are known (mass, shape, density, friction, mass distribution, etc.) [23]. In addition, this calculation often takes much longer than qualitative approaches and provides no guidance as to how to correct or adjust an unstable structure. Using a qualitative stability analysis approach allows us to estimate the stability of a structure much quicker, whilst sacrificing some accuracy.

Formal structure representation Based on our input structure we can construct a labelled directed graph where there is a node N_i for each block B_i within our structure and directed edges to specify supporting relations between two blocks. We call this the support graph (SG) of a structure. If the top horizontal edge of a block B_1 is touching the bottom horizontal edge of block B_2 (i.e. B_2 is resting on top of B_1), then SG contains an edge pointing from N_1 to N_2. For the sake of our definitions, the ground that a structure is resting on can simply be taken to be another, albeit very large, block (i.e. structure is resting on top of a ground block).

Definition 1. *(*Supporter, Support Depth, Supportees, Direct Supporter, Direct Supportees, Support Area*): Given a support graph SG, if there exists a path from N_i to N_j, then block B_i is a* supporter *of block B_j (B_i supports B_j).* Support depth $SD(N_i, N_j)$ *is the length of the shortest path from N_i to N_j. A* direct supporter *of a block B_j is a block B_i where $SD(N_i, N_j) = 1$. The* supportees *of block B_i is the set of all blocks that B_i is a* supporter *of. The* direct supportees *of block B_i is the set of all blocks that B_i is a* direct supporter *of. The* support area *for a block B_i is the horizontal interval between its leftmost and rightmost* direct supporters*.*

Using the example structure shown in Figure 2 (b) to help reinforce these definitions, Block C is a direct supporter of block D, an (indirect) supporter of block E, a direct supportee of block B, and an (indirect) supportee of block A. Each of these definitions can also be extended to apply to a collection of blocks rather than just a

single block. In this case the output is equal to the combined outputs when the definition is applied to each block within the collection, excluding blocks in the output that are themselves members of the collection being queried. (i.e. if $Q = [A, B, C]$, then $Supporters(Q) = [Supporters(A) \cup Supporters(B) \cup Supporters(C)] - Q$)

Prior qualitative methods There are currently two main qualitative methods which test for stability in 2D structures composed of multiple rectangles. The first method tests the stability of a structure by iteratively calculating the mass centre for a set of blocks from top to bottom, and checking if the vertical projection of this falls into the set of blocks' support area [20]. The second method determines structural stability by taking each block within the structure and its supportees as a substructure, and testing whether the vertical projection of its mass centre falls into the substructure's support area [14]. When applied to structures containing only axis-aligned blocks, it turns out that both methods perform exactly the same calculations but in a different order. These methods have a critical weakness however, in that all supporting blocks for the queried block's set of supportees are considered when calculating the supporting area. This assumption that all blocks in a set are supported equally by all supporting blocks often results in unstable structures being falsely classified as stable, such as the structure as shown in Figure 2 (b). In this example, block F is only a supporter of block E, but is also included when determining if blocks B, C and D are stable using these prior analysis methods.

Proposed algorithm We therefore propose a new qualitative stability test that is able to give a better approximation of stability compared to those previously described. For this algorithm, we assume that the densities of all blocks are uniformly distributed. This method does not produce perfect results, as qualitative approaches can only ever provide an estimate of stability, but is still able to detect the majority of unstable cases. This method also provides detailed feedback as to why a particular structure is unstable, allowing us to immediately adjust the structure to account for this. Algorithm 1 describes our proposed stability test (Note. The vertical projection of the mass centre is abbreviated to $VPMC$).

Unstable structure adjustment Based on the outcome of this stability test, we can adjust an unstable structure to make it stable. By ordering the blocks in our input structure based on the y-axis position of their mass centre, our improved stability algorithm will return both the highest unbalanced block (B is unbalanced at point P) and the side of that block (left or right) that has too much weight on it. An additional support block is then placed below either the left or right edge of this unbalanced block, depending on which side has too much weight. This added block's width is set to some default minimum value, and extends downwards until it reaches another block (or the ground). The stability of the new structure is then re-analysed, and this process repeats until the structure is deemed stable.

Example 1. Using the same structure from Figure 2 (b), we provide a step-by-step example to help explain our structure analysis / adjustment process:

Algorithm 1 Stability Test

1: **for all** B in $StructureBlocks$ **do**
2: $Z \leftarrow [B]$
3: $X \leftarrow [B \cup Supporters(B) \cup Supportees(B)]$
4: **for all** S in $Supportees(B)$ **do**
5: **if** all $Supporters(S)$ in X **then**
6: $Z \leftarrow Z \cup S$
7: **end if**
8: **end for**
9: **if** ($VPMC(Z)$ doesn't fall into $SupportArea(B)$ **then**
10: $P \leftarrow$ point in $SupportArea(B)$ closest to $VPMC(Z)$
11: **if** $VPMC(Z)$ is left of P **then**
12: $A \leftarrow$ area right of P
13: **end if**
14: **if** $VPMC(Z)$ is right of P **then**
15: $A \leftarrow$ area left of P
16: **end if**
17: **for all** N in $Supportees(B) \notin Z$ **do**
18: **if** N overlaps A **then**
19: $Z \leftarrow Z \cup N$
20: **end if**
21: **end for**
22: **if** ($VPMC(Z)$ doesn't fall into $SupportArea(Z)$ **then**
23: **if** $VPMC(Z)$ is left of P **then**
24: Return False ▷ B is unbalanced at point P (left)
25: **end if**
26: **if** $VPMC(Z)$ is right of P **then**
27: Return False ▷ B is unbalanced at point P (right)
28: **end if**
29: **end if**
30: **end if**
31: **end for**
32: Return True

– Our algorithm first checks the stability of block E. As block E has no supportees, the set Z simply contains block E (Z=[E]) (lines 2-8). The vertical projection of the mass centre of block E falls into its support area (horizontal interval between blocks D and F) (line 9) so this block is stable.
– Next we check the stability of block D. Block D has block E as a supportee, but as block E has a supporter that is not in X (block F), it will not be added to the set Z (Z=[D]) (lines 2-8). the vertical projection of the mass centre of block D falls into its support area (block C) (line 9) so this block is stable.
– Next we check the stability of block C. Block C has two supportees, blocks D and E. Block E is not added to the set Z for the same reason as before, but all supporters of block D are in X, so it is added to the set Z (Z=[C, D]) (lines 2-8). The vertical projection of the mass centre of the set of blocks [C, D] does not fall into the support area of block C (just block B) (line 9), so potentially this block is unstable.

P is set to the rightmost point in block B, and A is set to the area left of P (lines 10-16). None of the supportees of block C that aren't in Z (only block E in this case) overlap A, so Z remains unchanged (Z=[C, D]) (lines 17-21). As the vertical projection of the mass centre of Z does not fall into its support area (block B) (line 22), we conclude that the structure is unstable and that block C is unbalanced on the right side of point P (lines 23-28).

– Having determined both the highest unbalanced block (C) and the side of it with too much weight (right) we add an additional support block below the right edge of block C, see Figure 2 (c). The stability of this new structure is then re-analysed, but this time it is found to be stable.

2.3 Grouping Block Sets

Problem: *Define and identify known rules / relations between blocks or sets of blocks within a given structure based on their properties, that need to be satisfied during the generation process.*

Now that all blocks have been finalised, we can group blocks within the structure together based on their position, size and shape. This reasoning is not essential to the structure generation process, but can help to significantly improve its overall speed and accuracy by eliminating unfeasible or undesirable possibilities early when selecting block shapes. Two different systems are used to group similar blocks or sets of blocks together, referred to as the height grouping and shape grouping methods. Relations within each of these groupings are also transitive.

Height Grouping Rule: Two sets of blocks are in the same height group if they share both a direct supporter and a direct supportee. If two sets of blocks are in the same height group, then the combined heights of all blocks in each set must be the same. Using the same example from Figure 2 (b), we can use this rule to infer that the combined heights of blocks A, B, C and D, must be the same as the height of block F. By following this rule, we can significantly reduce the total runtime of our structure generation process by helping to detect unfeasible block shape combinations early when selecting block shapes for our final generated structure (used later in section 2.6).

Shape Grouping Rule: Two blocks ($B1$ and $B2$) are in the same shape group if the following conditions hold:

– $B1_{width} \approx B2_{width}$ (within set error percentage).
– $B1_{height} \approx B2_{height}$ (within set error percentage).
– $(B1_x \approx B2_x) \vee (B1_y \approx B2_y)$ (within set error percentage).
– There are no other blocks between $B1$ and $B2$.

(Note. the x-axis and y-axis location values for a block (B_x, B_y) are defined by its mass centre).

Any blocks within the same shape group must have the same block shape. The shape grouping rule is not as structurally important as the height grouping rule, but often leads to a final generated structure that is much closer to the original sketch (i.e. the shape grouping rule ensures that blocks in our input sketch which were intended by the drawer to be the same shape also have the same shape in the final generated structure).

2.4 Composite Blocks

Problem: *Generate additional composite block shapes within pre-defined size limits, given a collection of regular rectangular block shapes.*

As well as the regular block shapes that are available, it is also possible to combine multiple blocks together to create additional composite blocks with new dimensions. While initially similar in many regards to the rectangle packing problem [18, 9], the task of creating suitable composite blocks for 2D structures has many different considerations. Unlike traditional packing problems we do not have a limited number of blocks, only a limited number of block shapes. Our proposed process for creating different composite block options, within predefined limits on the maximum width $Width_{max}$ and height $Height_{max}$ that the block can have, is as follows:

Given a collection of N regular rectangular block shapes, sort them together into a set of groupings G based on their height. Remove from G any groupings that contain blocks with a height greater than $Height_{max}$. For each grouping G_k in G perform the following:

Identify all ordered combinations C_k of blocks within G_k, that when placed horizontally next to each other give a width less than $Width_{max}$. Each combination C_{ki} in C_k has three properties, the number of blocks within it $NumberBlocks(C_{ki})$, its total width $Width(C_{ki})$, and the locations of all *connection points* where one block ends and the next begins $ConnectPoints(C_{ki})$. Remove from C_k any combination C_{ki} if there exists any another combination C_{kj} where the following is true:

- $Width(C_{ki}) = Width(C_{kj})$
- $NumberBlocks(C_{ki}) > NumberBlocks(C_{kj})$
- $ConnectPoints(C_{ki}) \subset ConnectPoints(C_{kj})$

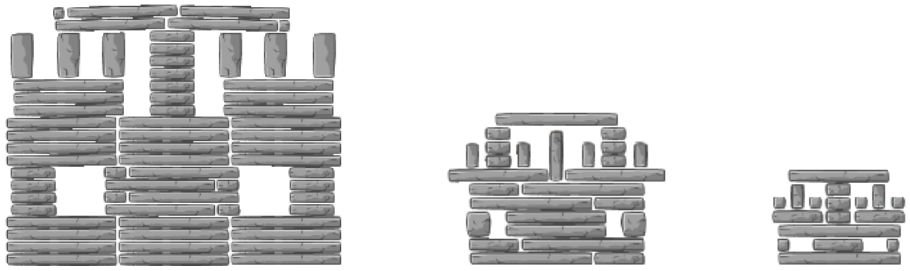
This removal process eliminates blocks combinations in C_k that are the same width as another combination, but are guaranteed to be equally or less structurally stable.

For each combination C_{ki} in C_k, perform the following:

1. $\mathcal{B} = C_{ki}$
2. $\mathcal{D}_{ki} \leftarrow \emptyset$
3. Add $\mathcal{B}$ to the set $\mathcal{D}_{ki}$
4. Reverse the order of the blocks in C_{ki}
5. Add C_{ki} as a new extra row of blocks on top of $\mathcal{B}$
6. If the height of $\mathcal{B}$ is less than $Height_{max}$, Go to step 3

This gives us a set of composite block shapes $\mathcal{D}_{ki}$ for each combination C_{ki} in C_k. Each of these $\mathcal{D}_{ki}$ sets can then be merged to give a combined set of composite block shapes $\mathcal{D}_k$ for all combinations in C_k. All $\mathcal{D}_k$ sets from each G_k grouping can then be merged to give a final set of additional composite block shapes $\mathcal{D}$, with dimensions not possible using regular block shapes alone. Comparing the generated structures in Figure 3 against the original rectangles in Figure 1 (h), demonstrates how multiple real blocks can be used to represent a single sketched block.

Note. In all subsequent sections, the term *block shapes* includes both regular and composite block shapes.



(a) $Scale_{max}$ (b) $Scale_{mid}$ (c) $Scale_{min}$

Fig. 3: Three example generated structures created from the same sketch in Figure 1, but using different scale calculations.

2.5 Block Scaling

Problem: *Scale a sketched structure so that it better fits the block shapes available.*

Another problem that must be solved before the final structure can be generated, is how to scale the sketched image such that the blocks within it are closer in size to the "real" block shapes available. If the input sketch is too small or too big, then the closest available real block is likely to always be the same. Without a fixed point of reference between the input sketch and the desired generated level, this problem has no perfect solution. We instead propose five different scale calculation options, the results of which can then be compared to determine the best approach:

- $Scale_{max} = Max(SBD)/Max(RBD)$
- $Scale_{min} = Min(SBD)/Min(RBD)$
- $Scale_{mid} = MidRange(SBD)/MidRange(RBD)$
- $Scale_{mean} = Mean(SBD)/Mean(RBD)$
- $Scale_{median} = Median(SBD)/Median(RBD)$

(SBD = sketched block dimension, RBD = real block dimension)

In more understandable terms, each scale calculation option associates one of the rectangle dimensions in the sketch with one of the real block dimensions available, i.e. using the $Scale_{max}$ calculation associates the largest rectangle dimension in our sketch with the largest real block shape dimension. Using each of these scale options often results in very distinctive generated structures with different block sizes and shapes, see Figure 3. These structures can then be ranked based on their similarity to the original sketch, with further details on this comparison procedure provided in the Structure Ranking section.

2.6 Selecting Block Shapes

Problem: *Given a sketched structure with rectangular blocks of any size / shape, generate a stable and feasible structure using our available block shapes that is similar in design to the original sketch*

Having described all the necessary components of our generator, we are now ready to start generating the final structure. Given a sketched structure $\mathcal{S}$ made of multiple rectangular blocks, we order the blocks using a bottom-up, depth first search algorithm (supporters always placed before the blocks they support). This block ordering $(\mathcal{S}_1, \mathcal{S}_2, ...\mathcal{S}_n)$ determines the order in which we select the block shapes for our final generated structure $\mathcal{G}$. To generate the ith block for $\mathcal{G}$, we first select the real block shape $Shape_i$ (including composite block shapes) that is closest to the shape of $\mathcal{S}_i$ (smallest non-overlapping region) and which hasn't already been tried for the current $\mathcal{G}$ definition before. A new block $\mathcal{G}_i$ with the shape of $Shape_i$ is then added to $\mathcal{G}$ in the same horizontal position as $\mathcal{S}_i$, and is vertically placed on top of its supporters determined by the support graph of $\mathcal{S}$ (due to our prior block ordering these will already have been added to $\mathcal{G}$). Five requirement checks are then carried out to make sure that $\mathcal{G}_i$'s shape and location are valid:

- R_1: $\mathcal{G}_i$ doesn't overlap another block in $\mathcal{G}$.
- R_2: $\mathcal{G}$ satisfies all grouping requirements of $\mathcal{S}$ (for both height and shape groupings).
- R_3: The support graph of $\mathcal{G}$ is consistent with the support graph of $\mathcal{S}$ (all blocks are supporters / supportees of those that they are supposed to be).
- R_4: Create a new structure $\mathcal{F}$, that contains all blocks currently in $\mathcal{G}$, as well as any blocks $\mathcal{S}_j$ in $\mathcal{S}$ where $\mathcal{G}_j$ is not in $\mathcal{G}$ (i.e. blocks already added to $\mathcal{G}$ use their real block shape, blocks that are not yet added to $\mathcal{G}$ use their sketched block shape). Run our previously described stability test on the structure $\mathcal{F}$, but using the support graph of $\mathcal{S}$. This stability test must return True (structure stable).
  (Note. even through the support graph of $\mathcal{S}$ may not match the support graph of $\mathcal{F}$, we can still use the support graph of $\mathcal{S}$ for determining supporters, supportees and support areas when performing our stability test on $\mathcal{F}$).
- R_5: If $Shape_i$ is a composite block shape, then all blocks that make up $\mathcal{G}_i$'s bottom row must be locally stable.

If any of these requirements $(R_1, R_2, R_3, R_4, R_5)$ are violated, then one of three possible adjustment options is performed:

- A_1: Move $\mathcal{G}_i$ horizontally either left or right by a small amount.
- A_2: Swap $Shape_i$ for the next closest block shape that has not already been tried for the current $\mathcal{G}$ definition before.
- A_3: Remove both $\mathcal{G}_i$ and $\mathcal{G}_{i-1}$ from $\mathcal{G}$ (i.e. backtracking).

After carrying out an adjustment (A_1, A_2, A_3) the structure requirements $(R_1, R_2, R_3, R_4, R_5)$ are re-tested. Adjustment A_1 is carried out first, in each direction for several distance values. Next, adjustment A_2 is carried out for several different alternative block shapes. Lastly, if the structure still does not satisfy our requirements after multiple shape changes and position shifts, then adjustment A_3 is performed. This process of selecting block shapes, testing structure requirements and performing adjustments, repeats recursively until either a final viable structure that satisfies all requirements is generated or all block shape combinations have been tested (structure generation not possible). Algorithm 2 provides a summative description of this block shape selection method.

Algorithm 2 Selecting Block Shapes

$GeneratedStructure \leftarrow \emptyset$
for all $Block$ in $SketchedStructure$ **do**
  $NewBlock \leftarrow Block$
  $Shape(NewBlock) \leftarrow ClosestBlockShape(Block)$
  add $NewBlock$ to $GeneratedStructure$
  $A1tries, A2tries \leftarrow 0$
  while any $(R_1, R_2, R_3, R_4, R_5)$ not satisfied **do**
    if $A1tries < A1tries_{max}$ **then**
      do adjustment A_1
    else if $A2tries < A2tries_{max}$ **then**
      do adjustment A_2
      $A1tries \leftarrow 0$
    else
      do adjustment A_3
      $A1tries, A2tries \leftarrow 0$
    end if
  end while
end for
Return $GeneratedStructure$

2.7 Structure Ranking

Problem: *Compare / rank different generated structures based on their similarity to the original sketch.*

As there are several different scaling options available, as well as other adjustable generation parameters, many different structures can usually be generated from the same sketch. Better results can often be achieved by generating multiple structures and then comparing them to determine which is best. This selection can be done manually based on user preference, but can also be done automatically using a similarity heuristic which measures how similar the generated structure is to the original sketch (after polygon splitting but before stability analysis). Four different measures of error are used in this heuristic calculation:

- $Error_{ratio}$ = Average percentage difference between each block's generated and sketched aspect ratios.
- $Error_{area}$ = Average difference between each block's generated and sketched areas.
- $Error_{position}$ = Average Euclidean distance between each block's generated and sketched locations, relative to the structure's centre of mass.
- $Error_{added}$ = Weighted sum of the areas of all blocks added during stability analysis.
- $SimilarityHeuristic = -(Error_{ratio} * Error_{area} * Error_{position}) - Error_{added}$

(Note. Both the sketched and generated structures are first scaled so that their total areas equal some arbitrary value).

Note that this similarity heuristic value is not normalised. In order to normalise this heuristic we would require a worst-case example to base the similarity value of -1 on, but it is not clear what a worst possible sketch would look like without setting some arbitrary bounds on the size and number of blocks for a generated structure.

A full quantitative test for stability is also conducted and if a structure is found to be unstable it is immediately rejected, thus guaranteeing that all generated structures are stable.

3 Evaluation

In order to evaluate our proposed generation algorithm, we investigated its ability to create levels for the video game Angry Birds based on human sketches. As previously mentioned, this game uses a suitable 2D physics engine and its levels often consist of one or more structures made from multiple rectangular blocks, with eight different block shapes available in the game. All experiments were performed on an Ubuntu 64-bit desktop PC with an i7-4790 CPU and 16GB RAM.

3.1 Experimental Results

Stability Analysis Comparison We first compared the accuracy of our new qualitative stability analysis method against the two main state-of-the-art techniques [20, 14]. This was done by generating 1000 random axis-aligned, rectilinear polygons using the approach described in [36]. Each of these polygons was then divided into rectangles using our polygon splitting algorithm and the subsequent structures analysed by all three stability methods. The exact stability of each generated structure was calculated using the algorithm described in [23]. Out of the 1000 polygons, 632 were stable whilst 368 were unstable. Neither our proposed method nor those previously described gave any false negatives (classified unstable but actually stable). However, both older methods each had 44 false positives (classified stable but actually unstable) whilst ours had only 18. This result indicates that our proposed stability analysis method performs significantly better than previous techniques when applied to our problem, and is able to accurately detect the vast majority of unstable structures. In all 350 cases where our stability analysis method correctly detected an unstable structure, it was also able to successfully adjust the structure to make it stable.

Similarity Heuristic Verification We also evaluated our proposed similarity heuristic to determine whether it provides a good measure of structural similarity between a sketch and a generated structure. We recruited 15 participants, 11 male and 4 female with an average age of 25.1, and asked each to draw 6 axis-aligned, rectilinear polygons of any design they liked using a simple pen and paper interface. These drawings were then scanned and our proposed generator used to create five different Angry Birds structures for each sketch, using our five different scaling calculations. These structures were then ranked by both the user and our similarity heuristic. The average Spearman's rank correlation coefficient over all 90 sketches was 0.834, indicating that our heuristic

value accurately measures perceived similarities between sketched and generated structures. Most participants were also extremely impressed by how accurately their original sketch could be represented within Angry Birds. The average generation time for each structure was 7.34 seconds and the average similarity heuristic value for the closest (best) generated structure was -21.55.

Recreating Original Angry Birds Levels To evaluate the overall performance of our entire generator, we investigated its ability to accurately replicate original levels from Angry Birds, based on human sketches of these same levels. Five different levels were selected from the "Poached Eggs" episode, specifically levels 1, 9, 13, 16, and 21, each of which contains a single complex structure. We collected sketches for each of these structures from 6 different participants, 4 male and 2 female with an average age of 22.8, giving a total of 30 structure sketches. Using our proposed similarity heuristic, we compared each sketch against both its closest generated structure and the original level it was based on. This allows us to compare how accurately our generation algorithm can replicate the sketched structure, compared to how closely the sketch resembles the original level. The average similarity value between each sketch and the level generated from it was -14.08, whilst the average similarity value between each sketch and the original level it was based on was -50.84. A breakdown of the average similarity heuristic scores for each level is displayed in Table 1. From these results we can see that our generator is able to replicate each sketched structure much closer than the average user can draw that same level.

Level #	1	9	13	16	21
Original	-58.4	-77.5	-23.6	-27.7	-67.0
Generated	-13.1	-15.9	-14.7	-8.58	-18.1

Table 1: Average similarity heuristic values when comparing human sketches against the original and generated structures.

The correlation coefficient for the similarity values between each sketch and both its original and generated structures is 0.477, indicating that there is a moderate positive relationship between these similarity scores for each level. This is probably because sketches that are further away from the original level are less likely to fulfil our generation requirements (e.g. overlapping blocks or unstable). Our generator will attempt to correct these issues by adjusting the structure, resulting in a worse similarity heuristic value. Figure 4 provides some examples from this experiment. The average generation time for each structure was 6.51 seconds.

When examining these results, please be aware that similarity heuristic values are only intended for comparing different generated structures based on the same input sketch to determine which is the closest, and should not be compared between different original structures (i.e. the similarity heuristic for a sketch based on a specific structure should not be compared against the similarity heuristic for a sketch based on a different structure).

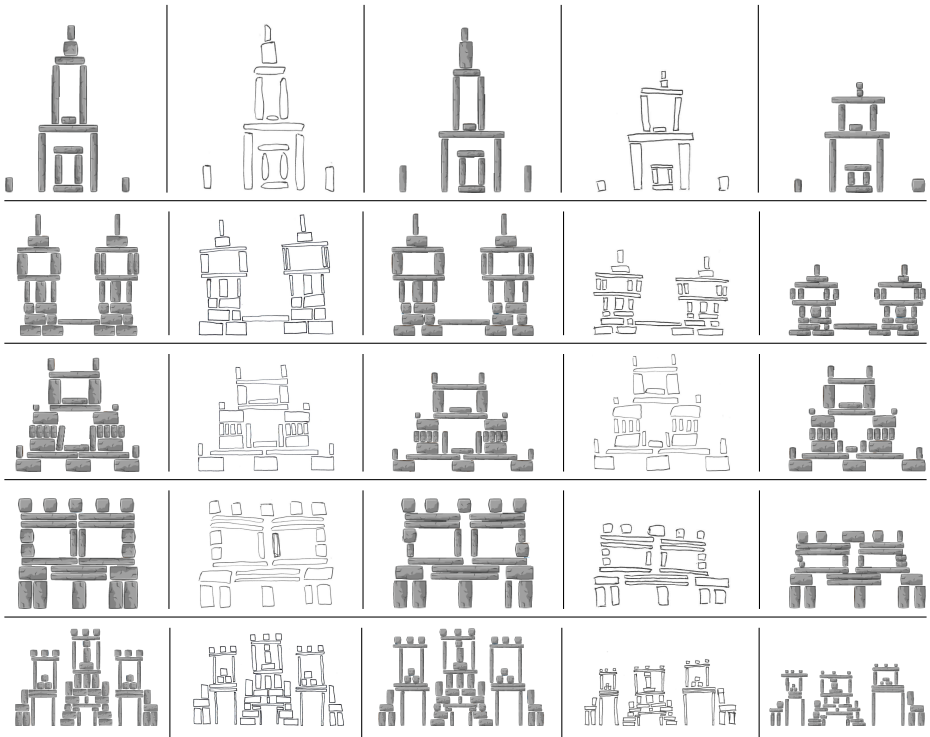


Fig. 4: The original structure (a), the best and worst human sketches (b)/(d), and the closest generated structures from these sketches (c)/(e)
Row 1 (level 1): $Similarity$(b,c) = -6.86, $Similarity$(b,a) = -14.69, $Similarity$(d,e) = -22.10, $Similarity$(d,a) = -92.39
Row 2 (level 9): $Similarity$(b,c) = -13.30, $Similarity$(b,a) = -17.62, $Similarity$(d,e) = -18.56, $Similarity$(d,a) = -127.35
Row 3 (level 13): $Similarity$(b,c) = -7.41, $Similarity$(b,a) = -19.55, $Similarity$(d,e) = -17.45, $Similarity$(d,a) = -39.48
Row 4 (level 16): $Similarity$(b,c) = -4.62, $Similarity$(b,a) = -16.26, $Similarity$(d,e) = -11.93, $Similarity$(d,a) = -64.24
Row 5 (level 21): $Similarity$(b,c) = -8.24, $Similarity$(b,a) = -12.35, $Similarity$(d,e) = -35.01, $Similarity$(d,a) = -82.79

3.2 Discussion and Future Work

The results of our evaluation demonstrate that our proposed generator can recreate both new and existing structures based solely on 2D human sketches, with a significant degree of accuracy. The spatial reasoning performed by our generator guarantees that all created structures are both stable and feasible within the required environment, whilst still ensuring that the users design is followed closely. Participants in our experiments

were able to use and understand the sketch-based interface easily, even if they had never previously played Angry Birds. Our methodology also possesses a large degree of flexibility, allowing for the incorporation of new requirements, desirable qualities or available block shapes when generating structures.

Outside of the obvious application to creating levels for physics-based games, this work has multiple other uses in a wide variety of different domains and situations. One potential example could be the possibility of a sketch-based interface for human-robot communication, that would allow users to intuitively explain how to complete complex physical tasks such as stacking items. The modularity of our method also allows specific sections to be improved or removed without significantly affecting others. Certain components of our generation process could be integrated with other already existing sketch-based interfaces for physics simulations, particularly those focused around cognitive science and education [13].

Future work for this research would naturally involve extending the range of possible structures that could be generated. Improvements to the generator might allow sketches to contain non-rectangular or angled blocks, and perhaps the ability to generate full 3D structures using technology such as stereoscopic displays and haptic interfaces [24]. These additions would require significant alterations to be made to both the stability and polygon splitting algorithms, as well as more advanced computer vision techniques for detecting multiple block shapes. Another more conceptual improvement would be to try and understand what certain users are actually attempting to represent in their sketched structures, rather than directly replicating what they draw.

4 Conclusions

In this paper, we have presented an approach to construct formal structure representations of rough human sketches using a limited number of rectangular block shapes, that accurately represents the original inputs while also ensuring that all physical requirements are satisfied. This combination of procedural content generation with sketch-based interfaces can help designers focus on what they want to create at a higher abstract level, without worrying about the physical requirements and limitations of the environment. This provides a way for inexperienced users to create their own content easily, whilst also allowing more experienced designers to rapidly construct prototypes for their ideas. With the huge surge in procedural content generation research over the past few years, it is not only feasible but also essential that more sophisticated ways to design virtual content are developed. We are confident that our proposed method represents a significant step forward in the task of allowing users to easily create personalised, complex and reliable digital content for physics-based environments, and presents a substantial contribution to the field of sketch-based and AI assisted content generation.

References

1. Alvarado, C., Davis, R.: SketchREAD: A multi-domain sketch recognition engine. In: Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology. pp. 23–32 (2004)
2. Anil Kumar, V.S., Ramesh, H.: Covering rectilinear polygons with axis-parallel rectangles. In: Proceedings of the Thirty-first Symposium on Theory of Computing. pp. 445–454 (1999)
3. Campos, C.R.F.G., de Oliveira Sa, W., Teixeira, J.M.G., Lelis, L.: Mixed-initiative tool to speed up content creation in physics-based games. In: Proceedings of SBGames 2017. pp. 590–593 (2017)
4. Cheema, S., LaViola, J.: PhysicsBook: A sketch-based interface for animating physics diagrams. In: Proceedings of the 2012 ACM International Conference on Intelligent User Interfaces. pp. 51–60. IUI '12 (2012)
5. Costa, A., Pereira, J.: SketchyDynamics: A library for the development of physics simulation applications with sketch-based interfaces. International Journal of Interactive Multimedia and Artificial Intelligence **2**(3), 23–30 (2013)
6. Costagliola, G., Rosa, M.D., Fuccella, V.: Rankfrag: A machine learning-based technique for finding corners in hand-drawn digital curves. In: International Conference on Distributed Multimedia Systems. pp. 29–38 (2015)
7. Davis, R.: Magic Paper: Sketch-understanding research. Computer **40**(9), 34–41 (2007)
8. Durocher, S., Mehrabi, S.: Computing partitions of rectilinear polygons with minimum stabbing number. In: Computing and Combinatorics. pp. 228–239 (2012)
9. E. Korf, R.: Optimal rectangle packing: New results. In: Proceedings of the 14th International Conference on Automated Planning and Scheduling. pp. 142–149 (2004)
10. Ferrari, L., Sankar, P., Sklansky, J.: Minimal rectangular partitions of digitized blobs. Computer Vision, Graphics, and Image Processing **28**(1), 58 – 71 (1984)
11. Ferreira, L.N., Toledo, C.: Tanager: A generator of feasible and engaging levels for Angry Birds. IEEE Transactions on Computational Intelligence and AI in Games (2017)
12. Field, M., Valentine, S., Linsey, J., Hammond, T.: Sketch recognition algorithms for comparing complex and unpredictable shapes. In: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence. pp. 2436–2441. IJCAI'11 (2011)
13. Forbus, K., Usher, J., Lovett, A., Lockwood, K., Wetzel, J.: CogSketch: Sketch understanding for cognitive science research and for education. Topics in Cognitive Science **3**(4), 648–666 (2011)
14. Ge, X., Renz, J., Zhang, P.: Visual detection of unknown objects in video games using qualitative stability analysis. IEEE Transactions on Computational Intelligence and AI in Games **8**(2), 166–177 (2016)
15. Gunther, O.: Minimum k-partitioning of rectilinear polygons. Journal of Symbolic Computation **9**(4), 457 – 483 (1990)
16. Hammond, T., Davis, R.: Tahuti: a geometrical sketch recognition system for UML class diagrams. In: SIGGRAPH (2006)
17. Hendrikx, M., Meijer, S., Velden, J.V.D., Iosup, A.: Procedural content generation for games: A survey. Trans. Multimedia Comput. Commun. Appl. **9**(1), 1–22 (2013)
18. Huang, E., Korf, R.E.: New improvements in optimal rectangle packing. In: Proceedings of the 21st International Jont Conference on Artifical Intelligence. pp. 511–516 (2009)
19. Imai, H., Asano, T.: Efficient algorithms for geometric graph search problems. SIAM Journal on Computing **15**(2), 478–494 (1986)
20. Jia, Z., Gallagher, A., Saxena, A., Chen, T.: 3D-based reasoning with blocks, support, and stability. In: IEEE Conference on Computer Vision and Pattern Recognition (2013)

21. Johnston, A., Carneiro, G., Ding, R., Velho, L.: 3-D modeling from concept sketches of human characters with minimal user interaction. In: International Conference on Digital Image Computing: Techniques and Applications (DICTA). pp. 1–8 (2015)
22. Liapis, A., Yannakakis, G.N., Togelius, J.: Sentient Sketchbook: Computer-aided game level authoring. In: Proceedings of the 8th Conference on the Foundations of Digital Games. pp. 213–220 (2013)
23. M. Blum, A.G., Neumann, B.: A stability test for configurations of blocks. Tech. rep., Massachusetts Institute of Technology (1970)
24. Onkar, P., Sen, D.: Controlled direct 3d sketching with haptic and motion constraints. International Journal of Computer Aided Engineering and Technology **8** (2016)
25. O'Rourke, J.: Uniqueness of orthogonal connect-the-dots. Machine Intelligence and Pattern Recognition **6**, 97–104 (1988)
26. O'Rourke, J., Tewari, G.: The structure of optimal partitions of orthogonal polygons into fat rectangles. Computational Geometry **28**(1), 49 – 71 (2004)
27. Ouyang, T.Y., Davis, R.: Recognition of hand drawn chemical diagrams. In: Proceedings of the 22Nd National Conference on Artificial Intelligence. pp. 846–851. AAAI'07 (2007)
28. Renz, J., Ge, X., Verma, R., Zhang, P.: Angry Birds as a challenge for artificial intelligence. In: Proceedings of the 30th AAAI Conference. pp. 4338–4339 (2016)
29. Shi, J., Tomasi, C.: Good features to track. In: 1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. pp. 593–600 (1994)
30. Shpitalni, M., Lipson, H.: Classification of sketch strokes and corner detection using conic sections and adaptive clustering. ASME Journal of Mechanical Design **119** (2001)
31. Smelik, R., Tutenel, T., de Kraker, K., Bidarra, R.: A declarative approach to procedural modeling of virtual worlds. Computers & Graphics **35**(2), 352 – 363 (2011)
32. Smith, G., Whitehead, J., Mateas, M.: Tanagra: A mixed-initiative level design tool. In: Proceedings of the Fifth International Conference on the Foundations of Digital Games. pp. 209–216. FDG '10 (2010)
33. Snodgrass, S., Ontañón, S.: Controllable procedural content generation via constrained multi-dimensional markov chain sampling. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence. pp. 780–786. IJCAI'16, AAAI Press (2016)
34. Stephenson, M., Renz, J.: Generating varied, stable and solvable levels for Angry Birds style physics games. In: 2017 IEEE Conference on Computational Intelligence and Games (CIG). pp. 288–295 (2017)
35. Togelius, J., Yannakakis, G.N., Stanley, K.O., Browne, C.: Search-based procedural content generation: A taxonomy and survey. IEEE Transactions on Computational Intelligence and AI in Games **3**(3), 172–186 (2011)
36. Tomás, A.P., Bajuelos, A.L.: Quadratic-time linear-space algorithms for generating orthogonal polygons with a given number of vertices. In: Computational Science and Its Applications – ICCSA. pp. 117–126 (2004)
37. Turquin, E., Cani, M.P., Hughes, J.F.: Sketching garments for virtual characters. In: SIGGRAPH (2007)
38. Wolin, A., Paulson, B., Hammond, T.: Sort, merge, repeat: An algorithm for effectively finding corners in hand-sketched strokes. In: Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling. pp. 93–99 (2009)
39. Xiong, Y., LaViola, Jr., J.J.: Revisiting ShortStraw: Improving corner finding in sketch-based interfaces. In: Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling. pp. 101–108. SBIM '09, ACM, New York, NY, USA (2009)
40. Zhang, P., Renz, J.: Qualitative spatial representation and reasoning in Angry Birds: The extended rectangle algebra. In: Knowledge Representation and Reasoning Conference (2014)