Scene Synthesis with Automated Generation of Textual Descriptions (Supplementary Material)

J. Müller-Huschke, M. Ritter*, and M. Harders

Interactive Graphics and Simulation Group at the Department of Computer Science, University of Innsbruck, Austria.
*corresponding author

1. Scene evaluation function

To build the evaluation function from the set of constraints the approach of [YYW*12] is utilized. In their work they build a function that scores a scene layout between 0 and 1 by taking the product of many factors. Each factor represents a logical connective. So, if constraints on the scene layout can be formulated with logical connectives, this method allows to easily build an evaluation function from constraints. [YYW*12] define these logical connective factor functions based on the Gaussian density function $\mathcal{N}(x,\mu,\sigma^2)$ and a sigmoid function $\mathrm{Sig}(x,h) = \frac{1}{1+e^{-hx}}$:

$$\begin{aligned} \mathrm{Equals}(x,y,\sigma^2) &= \mathcal{N}(0,\|x-y\|,\sigma^2)/\mathcal{N}(0,0,\sigma^2) \\ \mathrm{Greater}(x,y,h) &= \mathrm{Sig}(x-y,h) \\ \mathrm{Less}(x,y,h) &= \mathrm{Sig}(y-x,h) \\ \mathrm{Range}(x,y_{\min},y_{\max},h) &= \mathrm{Greater}(x,y_{\min},h)\mathrm{Less}(x,y_{\max},h). \end{aligned} \tag{1}$$

The constants h and σ^2 define their steepness. In this work they are chosen the same as in [YYW*12], i.e. $h = 3.0$ and $\sigma^2 = 0.1$. These logical constraints can be used to define spatial constraints. For example, if a scene contains two objects A and B, which should be at a perpendicular angle to each other and two meters apart from each other, the scene evaluation function could look like this:

$$\begin{aligned} f = &\mathrm{Equals}(\mathit{angleBetween}(A,B),90.0,\sigma^2)\cdot \\ &\mathrm{Greater}(\mathit{distanceBetween}(A,B),2.0,h). \end{aligned} \tag{2}$$

The evaluation function takes scene layout as input and produces an evaluation score between 0 and 1, with values close to 1 representing the condition that all constraints are perfectly satisfied. Ideally, no scene layout should ever be scored with 0, since then calculating the acceptance ratio for Metropolis Hastings becomes undefined. Additionally, it is likely that neighboring scene layouts in the search space will also be scored with 0, removing any sort of gradient that could allow the Markov chain to quickly converge. The evaluation function f is defined and computed in log space using an energy function v. This is in order to avoid underflows, e.g. the evaluation function becoming 0 due to floating point precision limitations when taking the product of many very small numbers. In log space these products become sums mitigating the issue to some extent.

$$f = e^{-v}. \tag{3}$$

The energy function v consists of four parts concerning different types of constraints:

$$v = v_r + v_c + v_p + v_q. \tag{4}$$

- v_r results from spatial relationships between objects.
- v_c states objects should not collide with one another.
- v_p states objects should not collide with pathways.
- v_q encourages generation of believable pathways.

For each edge of the scene relationship graph, an evaluation factor is computed and summed up to form v_r. This factor is built from one or multiple factor functions from [YYW*12] (see equation 1). However, these factor functions are usually instantiated in log space, which is denoted here by writing their names in lower case. Table 1 shows how to compute the evaluation factors for all possible types of edges in the scene relationship graph. To keep this table compact, an additional factor function enforces that two unit vectors $\vec{v},\vec{w}$ point in the same direction:

$$\textbf{pointingSameDirection}(\vec{v},\vec{w}) = \textbf{equals}(\cos^{-1}(\vec{v}\cdot\vec{w}),0). \tag{5}$$

A number of functions are used in table 1 in order to get information about the relative transformation between scene objects and their collision geometries:

- $\mathit{distanceBetween}(A,B)$ gives the minimum distance that can be found between the collision geometries of objects A and B.
- $\mathit{directionFromTo}(A,B)$ gives the normalized direction vector pointing from the position of A to the position of B.
- $\mathit{direction}(A,\vec{x})$ gives the direction vector $\vec{x}$ transformed to the local coordinate system of object A.
- $\mathit{boundary}(A,\vec{x})$ gives the most extreme point in direction $\vec{x}$ on the collision geometry of A. The resulting point is given in the global coordinate system.
- $\mathit{distanceToBoundary}(A,B,\vec{x})$ gives the distance between the collision geometry of A and the boundary in $\vec{x}$ direction of the collision geometry of B. For convex collision shapes this boundary would most likely be a point, but for box colliders it can be a plane.

Edge Label from A to B	**Resulting factor**
NextTo NextToChooseDirection	$\textbf{less}(\textit{distanceBetween}(A,B),2)$
NextToChooseSide	$\textbf{less}(\textit{distanceBetween}(A,B),2)+\max($ $\textbf{pointingSameDirection}(\textit{directionFromTo}(B,A),\textit{direction}(B,\text{left})),$ $\textbf{pointingSameDirection}(\textit{directionFromTo}(B,A),\textit{direction}(B,\text{right})))$
NextToNorth	$\textbf{less}(\textit{distanceBetween}(A,B),2)+$ $\textbf{pointingSameDirection}(\textit{directionFromTo}(B,A),\text{north})$
NextToEast	$\textbf{less}(\textit{distanceBetween}(A,B),2)+$ $\textbf{pointingSameDirection}(\textit{directionFromTo}(B,A),\text{east})$
NextToSouth	$\textbf{less}(\textit{distanceBetween}(A,B),2)+$ $\textbf{pointingSameDirection}(\textit{directionFromTo}(B,A),\text{south})$
NextToWest	$\textbf{less}(\textit{distanceBetween}(A,B),2)+$ $\textbf{pointingSameDirection}(\textit{directionFromTo}(B,A),\text{west})$
NextToLeft	$\textbf{less}(\textit{distanceBetween}(A,B),2)+$ $\textbf{pointingSameDirection}(\textit{directionFromTo}(B,A),\textit{direction}(B,\text{left}))$
NextToRight	$\textbf{less}(\textit{distanceBetween}(A,B),2)+$ $\textbf{pointingSameDirection}(\textit{directionFromTo}(B,A),\textit{direction}(B,\text{right}))$
NextToFront	$\textbf{less}(\textit{distanceBetween}(A,B),2)+$ $\textbf{pointingSameDirection}(\textit{directionFromTo}(B,A),\textit{direction}(B,\text{forward}))$
NextToBehind	$\textbf{less}(\textit{distanceBetween}(A,B),2)+$ $\textbf{pointingSameDirection}(\textit{directionFromTo}(B,A),\textit{direction}(B,\text{backward}))$
OnTopOf IsPartOf	$\textbf{equals}(\textit{boundary}(A,\text{up})_y,\textit{boundary}(B,\text{down})_y)+$ $\textbf{range}(\text{center}(A)_x,\textit{boundary}(B,\text{left})_x,\textit{boundary}(B,\text{right})_x)+$ $\textbf{range}(\text{center}(A)_z,\textit{boundary}(B,\text{forward})_z,\textit{boundary}(B,\text{backward})_z)$
OnTopOfMiddle IsPartOfMiddle	$\textbf{equals}(\textit{boundary}(A,\text{up})_y,\textit{boundary}(B,\text{down})_y)+$ $\textbf{equals}(\lVert\text{center}(A)_{xz}-\text{center}(B)_{xz}\rVert,0)$
OnTopOfEdge IsPartOfEdge	$\textbf{equals}(\textit{boundary}(A,\text{up})_y,\textit{boundary}(B,\text{down})_y)+\textbf{equals}(\min($ $\textit{distanceToBoundary}(A,B,\text{forward}),\textit{distanceToBoundary}(A,B,\text{backward})$ $\textit{distanceToBoundary}(A,B,\text{left}),\textit{distanceToBoundary}(A,B,\text{right}),0)$
OnTopOfFrontEdge IsPartOfFrontEdge	$\textbf{equals}(\textit{boundary}(A,\text{up})_y,\textit{boundary}(B,\text{down})_y)+$ $\textbf{equals}(\textit{distanceToBoundary}(A,B,\text{forward}),0)$
FacingTowards ImplicitFacingTowards	$\textbf{pointingSameDirection}(\textit{directionFromTo}(A,B),\textit{direction}(A,\text{forward}))$
FacingAwayFrom ImplicitFacingAwayFrom	$\textbf{range}(\textit{direction}(A,\text{forward})\cdot\textit{direction}(B,\text{forward})),-1,0)$
FacingSameDirection ImplicitFacingSameDirection	$\textbf{pointingSameDirection}(\textit{direction}(A,\text{forward}),\textit{direction}(B,\text{forward}))$
FacingOrthogonalDirection ImplicitFacingOrthogonalDirection	$\textbf{equals}(\cos^{-1}(\textit{direction}(A,\text{forward})\cdot\textit{direction}(B,\text{forward}))),\frac{\pi}{2})$
PathConnectedTo	0

Table 1: *Translation of edge types from the scene relationship graph to evaluation factors in log space. Each edge points from an object A to an object B. Factor functions are printed in bold. They are log space versions of the functions [YYW* 12] employed (see Equation 1).*

For these functions no further implementation is detailed, since they are highly dependent on the collision geometries used.

Objects in the scene should not collide with each other. The function v_c penalizes collisions between each object pair from the set of objects in the scene O and can be written as:

$$v_c = \sum_{(A,B)\in O\times O, A\neq B} collisionPenalty(A,B)$$
$$collisionPenalty(A,B) = \begin{cases} \log((1-penetrationDepth(A,B))^c) & \\ \qquad\qquad \text{if } A \text{ and } B \text{ collide} & \\ 0 \qquad\qquad \text{otherwise.} & \end{cases} \tag{6}$$

The function $penetrationDepth(A,B)$ calculates the percentage that the collision geometries of objects A and B penetrate into each other. In other words, $penetrationDepth(A,B)$ returns the minimum distance that one of the two objects needs to move so that their collision geometries do not intersect anymore, normalized to $[0,1]$ using the summed up size of both objects. An intensity constant c controls how much more large intersections should be penalized compared to small intersections. Since $penetrationDepth(A,B) \in [0,1]$, a larger c will penalize big penetration depths much more. The choice of c additionally affects, which priority the avoidance of collisions should have relative to other constraints.

The paths in the scene layouts are made up of path segments with control points, v_p and v_q of the evaluation function are defined to yield reasonable path layouts. Given a set of path segments P and objects O, v_p penalizes collisions between path segments and objects similar to v_c:

$$v_p = \sum_{A\in O, B\in P} collisionPenalty(A,B). \tag{7}$$

Here, the function $collisionPenalty(A,B)$ is defined the same way as in equation 6 above. v_q encourages each path segment from P to be in a specific shape. For each path segment $A \in P$ consisting of n_A control points, $C_{A,i}$ with $1 < i < n_A$ denotes the individual control points of the segment and l_A the desired maximum length; v_q is defined as:

$$\begin{aligned} v_q = &\sum_{A\in P} \mathbf{less}(totalLength(A), l_A) + \\ &\sum_{A\in P} \sum_{1<i<n_A} \mathbf{less}(angleBetween(C_{A,i-1}, C_{A,i}, C_{A,i+1}), \gamma) + \\ &\sum_{\alpha\in anglesIntersectionPaths(P)} \mathbf{greater}(\alpha,\gamma). \end{aligned} \tag{8}$$

The factors applied in v_q can be explained the following way:

- $\sum_{A\in P} \mathbf{less}(totalLength(A), l_A)$ encourages the path segment to be shorter than its assigned maximum length, with $totalLength(A)$ summing up the length of all lines the path segment draws.
- $\sum_{A\in P}\sum_{1<i<n_A} \mathbf{less}(angleBetween(C_{A,i-1}, C_{A,i}, C_{A,i+1}), \gamma)$ penalizes angles which are bigger than some constant γ (here chosen to be $60°$). The angles are between lines connecting successive control points $C_{A,i-1}, C_{A,i}, C_{A,i+1}$.
- $\sum_{\alpha\in anglesIntersectionPaths(P)} \mathbf{greater}(\alpha,\gamma)$ makes all outgoing paths segments of an intersection object have at least a 60 degree angle between them. A function with the name $anglesIntersectionPaths(P)$ returns the set of all angles between neighboring outgoing paths segments from all intersection objects. For scenes without path intersections it yields an empty set.

Notably missing from constraints v_q and v_c is collision between path segments. This is acceptable, since given the above limitations on path segment lengths and angles, collisions of path segments can only happen rarely unless many path segments in the scene have a large number of control points.

2. Additional Details about the Pilot User Study

Participants were informed about the purpose of the study and gave their informed consent. They received no financial compensation and the study was conducted online. The developed web interface allowed to explore the scenes by rotating the camera and zooming in and out.

At the beginning of the questionnaire, subjects were asked to assess on a Likert scale their familiarity with the overall project as well as with 3D virtual worlds (video games, simulations) in general. No significant correlation between these self assessments and the later study results were found. An optional post-hoc questionnaire allowed participants to rate difficulty and clarity of the study, and to provide remarks in a free-form text field. Due to a technical mishap, in a few rare cases participants were able to rate descriptions they themselves provided in the previous session. Nevertheless, excluding these cases did not change significance of the results or other findings.

3. Scene Variety, Model Database and Further examples

To generate a wide variety of scene layouts 250 models were obtained from the Unity Asset Store (https://assetstore.unity.com/packages/3d/environments/fantasy/fantasy-village-pack-140303) and annotated with metadata to populate the scenes. For the graph grammar, over 100 different production rules were hand-designed in about 30 person-hours.

Further examples of generated scenes with descriptions demonstrating some of the variety and complexity possible are presented below. For each scene the automated description, the highest rated and the lowest rated user description is shown. Examples where the developed method worked well are given in Figures 1 and 2. An example where both the scene layout and the the textual description exhibit major flaws is shown in Figure 3. Failure cases are presented in Figures 4 and 5.

References

[YYW*12] YEH Y.-T., YANG L., WATSON M., GOODMAN N. D., HANRAHAN P.: Synthesizing open worlds with constraints using locally annealed reversible jump MCMC. *ACM Trans. on Graphics 31*, 4 (July 2012). 1, 2

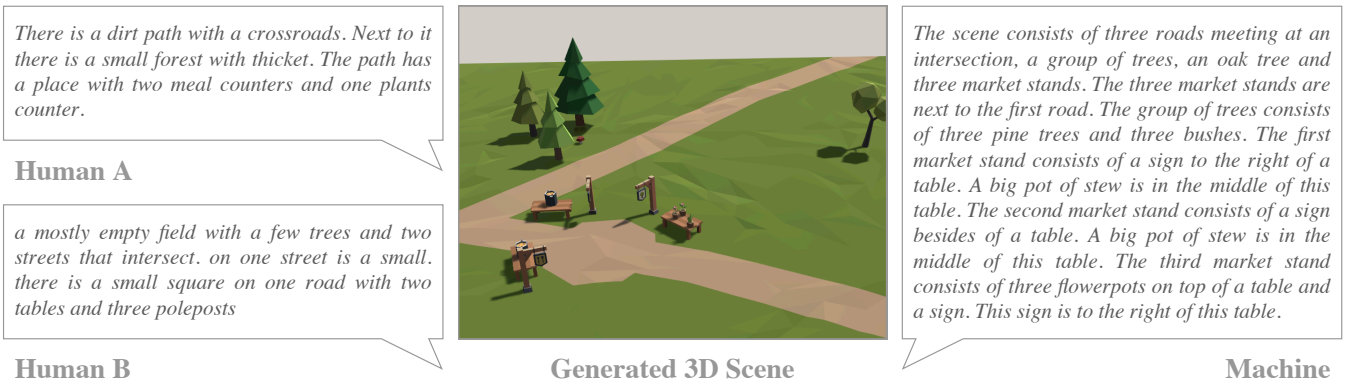

Figure 1: *Procedurally generated 3D scene with textual descriptions. (Left:) Two descriptions by human users – rated as most (top) and least (bottom) accurate in our pilot study; (Right:) automatically generated description with our framework.*

Figure 2: *Procedurally generated 3D scene with textual descriptions. (Left:) Two descriptions by human users – rated as most (top) and least (bottom) accurate in our pilot study; (Right:) automatically generated description with our framework.*

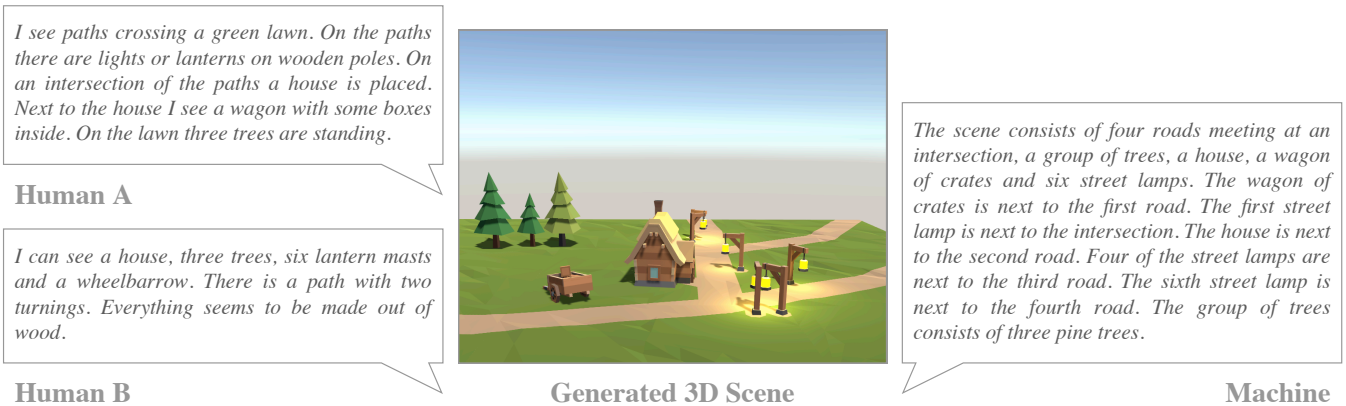

Figure 3: *Procedurally generated 3D scene with textual descriptions with major flaws. The scene layout displays too many street lanterns placed chaotically and the description incorrectly describes the presented trail layout. (Left:) Two descriptions by human users – rated as most (top) and least (bottom) accurate in our pilot study; (Right:) automatically generated description with our framework.*

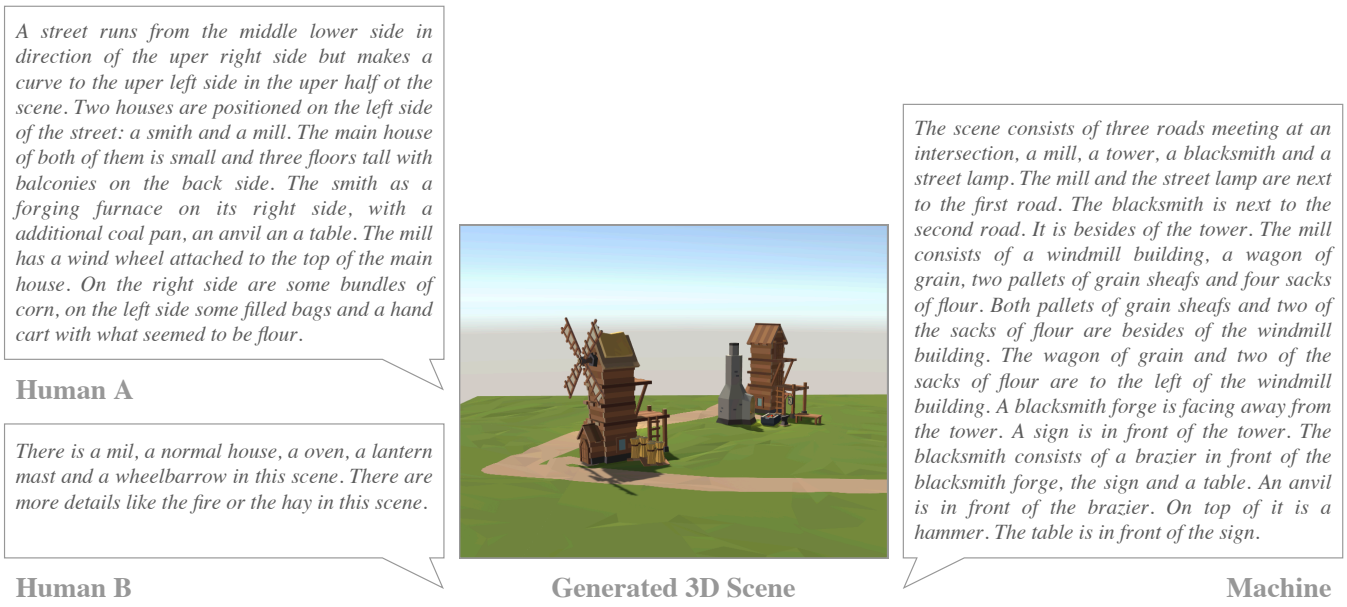

Figure 4: *Procedurally generated 3D scene example with textual descriptions. The generated trails in the scene overlap each other, resulting in a scene with an incorrect description. (Left:) Two descriptions by human users – rated as most (top) and least (bottom) accurate in our pilot study; (Right:) automatically generated description with our framework.*

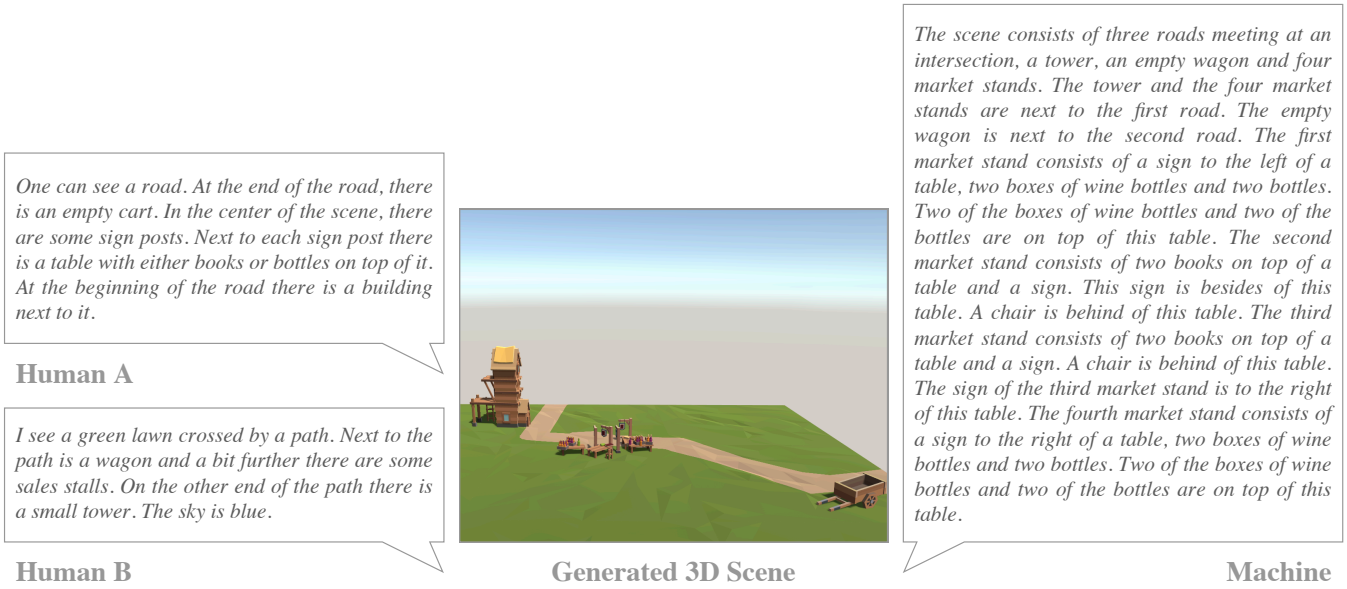

Figure 5: *Procedurally generated 3D scene example with textual descriptions. Many generated objects and the trails clip into each other, resulting in a not plausible scene with an incorrect description. (Left:) Two descriptions by human users – rated as most (top) and least (bottom) accurate in our pilot study; (Right:) automatically generated description with our framework.*