Layout Design for Augmented Reality Applications

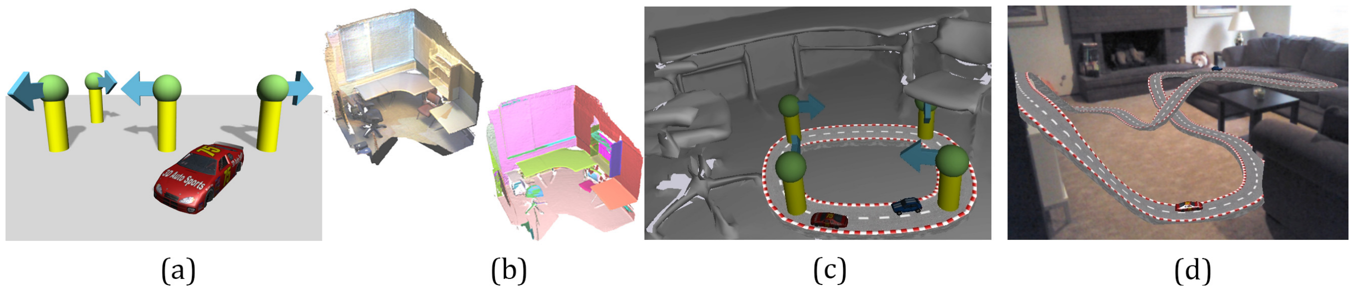

Figure 1: *Designing an immersive augmented reality (AR) application such as a dynamic racing game is difficult. In our framework (a) declarative rules are used to define game objects and the rules governing them (b) in real-time we analyze an environment to extract scene geometry and horizontal and vertical planes (c) our exponential move-making algorithm targets the application to the room (d) an additional result of our system in a different room with a much longer track length.*

Abstract

Creating a layout for an augmented reality (AR) application which embeds virtual objects in a physical environment is difficult as it must adapt to any physical space. We propose a rule-based framework for generating object layouts for AR applications. We present an algorithm for dynamically targeting the AR application to a new environment in real time by solving a constraint-satisfaction problem. Under our framework, the developer of an AR application specifies a set of cost functions (rules) which enforce self-consistency (rules regarding the inter-relationships of application components) and scene-consistency (application components are consistent with the physical environment they are placed in). Our method is general and can be applied to any rule-based layout design problems. We represent layout rules using hyper-graphs where nodes in the graph represent objects and hyper-edges between nodes represent the rules that operate on objects.

Given an environment, we create a layout for an application using a novel solution-space exploration algorithm. Our method exploits the fact that for many types of rules, satisfiable assignments can be found efficiently, in other words, these rules are *locally* satisfiable. This allows us to sample candidate object values from the known partial probability distribution function for each rule. Experimental results demonstrate that this sampling technique reduces the number of samples required by other algorithms by orders of magnitude enabling us to find rule-consistent augmentations for the scene. We demonstrate several augmented reality applications, which automatically adapt to different rooms and changing circumstances in each room. Our adaptive search algorithm is general and can be used for many other applications such as automatic furniture layout, populating virtual worlds and 2D graphic design.

CR Categories: F.4.1 [Mathematical Logic]: Logic and Constraint Programming— [G.3]: Probability and Statistics—Markov Processes;

Keywords: weighted constraint optimization, layout synthesis, augmented reality

1 Introduction

Augmented reality is a growing trends both on mobile platforms, as well as on emerging wearable computing platforms. Yet, AR systems have struggled to make the transition from laboratory to the real world. A particular hindrance to the successful deployment of AR systems is the complex and variant nature of reality. AR apps must work in any environment the user finds herself in. Therefore, the layout of the different elements comprising the AR application must be consistent with the environment. A particular issue that makes this task challenging is the fact that layout of virtual objects must be both *self-consistent*, i.e. consistent with the placement of other virtual objects, as well as *scene-consistent*, i.e. consistent with the geometry of the physical environment they are placed in. For example an application might require that two game objects be placed within two feet of each other (self-consistent) but also they be placed on an elevated horizontal surface (scene-consistent).

We describe FLARE (Fast Layout for Augmented Reality), a system which enables targeting AR applications to a variety of environments. An AR application is designed using declarative rules, describing the correct mapping of the application to an environment.

We capture the user's current environment using a Kinect camera (rgb and depth streams), and process it using Kinect Fusion [Newcombe et al. 2011] to extract dense scene geometry. We further process the scene to detect planar surfaces in the room and label them as vertical (e.g. walls) or horizontal (e.g. floor, table). Planar features are common in indoor scenes and are useful to many applications. Adding additional detectors (e.g. object detection, recognizing previously visited rooms) can enable more complex rules and applications. FLARE performs a real-time mapping of the application to the user's current environment, by applying the rules to the application objects, and the geometric info extracted from the scene.

Using the declarative rules, we formulate virtual object placement as a weighted constraint-satisfaction problem. Our formulation incorporates both low-order and high-order interactions between design elements. Low-order rules are defined over one or two design elements. For example a rule that states that an object must be placed on a vertical surface (such as a wall). On the other hand, high-order interactions are defined over large number of design elements and can capture higher-order relationships between objects like co-linearity, co-planarity, equidistant, that are important for effectively augmenting the real scene with virtual objects. Note that this approach extends easily to optimizing other object properties such as material (color, texture).

We represent the rules that need to be satisfied by the objects using hyper-graphs where nodes in the graph represent objects and hyper-edges between nodes represent the rules that operate on objects.

Computation of the optimal solution under a given hyper-graph requires minimization of a non-convex function, which in general, is infeasible. A number of approximate methods have been proposed in the literature but even they are computationally expensive for graphs with arbitrary higher-order relationship (hyper-edges).

At the core of our system is an algorithm which takes as input a set of rules, and encodes the resulting hyper-graph in a simple graph in which each node represents an object (and its properties) or an auxiliary node (used to represent complex relationships). The rules define cost functions on the nodes and edges of the graph. The algorithm, in each iteration, generates candidate values for each object, and evaluates them simultaneously to find the approximate best layout (given the current set of candidates). The key idea that drives our algorithm is an observation that for many types of rules, satisfiable assignments can be found efficiently. In other words, these rules are locally (individually) satisfiable i.e. we can generate proposals for objects that locally satisfy individual rules and reduce the need for blind sampling.

Experimental results demonstrate that our *locally consistent* sampling technique is very efficient and requires substantially fewer number of samples compared to other algorithms. Apart from the AR motivated object placement in 3D scenes problem, we also show the applicability of our approach to furniture arrangement (comparing to previous work) and both geometric and photometric 2D targeting problems (in supplemental material).

Our contributions are (1) FLARE, a general framework for designing the layout of an AR application (2) a quick-converging algorithm for finding an optimal layout, geared towards low-powered computing devices.

The rest of the paper is organized as follows, in section 2 we discuss related work. In section 3 we provide a formalization for the layout design problem and describe how rules operating on objects can be represented using graphs. In section 4 we describe our method for generating compact mathematical descriptions of design rules and our algorithm for computing the optimal layout. In section 5 we provide details of the experimental evaluation and show qualitative and quantitative results. We conclude in section 6 by listing some observations regarding our framework and discussing directions for future work.

2 Related Work

Mapping AR to the real world Augmented reality [Azuma et al. 2001; Carmigniani et al. 2011], in general, should work in a large range of environments. Different approaches were used in the past starting with simply ignoring the structure of the world. Mobile AR application such as [Layar 2013; Wikitude 2013] use the location of the user and the orientation of the mobile device to add a 2D overlay over the user's view. For location-specific apps, the geometry of a site can be computed in advance, for example archaeological sites [Architip 2013], Museums [Margriet Schavemaker and Pondaag 2011], manufacturing floor [Ong and (Eds.) 2004], projection mapping [Grasset et al. 2003]. In recent years many augmented reality apps and games, were designed for a simple planar world on which the augmented content resides. The world plane is attached to a known pattern, found on a magazine ad or a packaging of a product [Layar 2013; Junayo 2013]. Recent works [Newcombe et al. 2011; Jones et al. 2013] used the recovered 3D geometry of the scene to demonstrate some physical simulation examples.

Layout Synthesis The availability of 3D models of physical spaces has inspired a large amount of work on generating layouts. In [Yu et al. 2011; Merrell et al. 2011] a set of rules and spatial relationships for optimal furniture positioning are established from examples and expert-based design guidelines. These rules are then enforced as constraints to generate furniture layout in a new room. [Yu et al. 2011] employed a simulated annealing method which is effective but takes several minutes, while [Merrell et al. 2011] sample a density function using the Metropolis-Hastings algorithm implemented on a GPU. They evaluate a large number of assignments and achieve interactive rates (requiring a strong GPU). Both papers work with a small number of objects in relatively small rooms and in static scenarios. [Fisher et al. 2012] showed how arrangements of 3D objects can be found using a data-driven example based approach. [Yeh et al. 2012] populate a scene with a variable number of objects (open universe). They present a probabilistic inference algorithm extending simulated annealing with local steps, however the computation cost is high and the procedure takes upwards of 30 minutes.

Constraint Satisfaction for Design The problem of rule based design generation has a long history. Design and layout synthesis consist of rules referencing a set of objects. An assignment to each object can be measured by how well the rules are met, whether they are satisfied or violated. As an example of an early work in this space, the Ultraviolet [Borning and Freeman-Benson 1998] system used a constraint satisfaction algorithm framework for interactive graphics. The constraints for user interface layout usually form a non-cyclic graph, are hierarchical in nature and container based and are therefore less complex.

Constraint satisfaction problems (CSP) [Mackworth 1977] are fundamental in Artificial Intelligence and Operations Research. A variant of the problem, weighted CSP defines a cost function assigned to each constraint, and the objective is to minimize the overall cost. A large majority of CSP algorithms [Kumar 1992] use a search paradigm over a limited set of possible object assignments. More recently, [Lin et al. 2013] used rules represented as factor graph to perform Pattern Colorizations. These approaches are relatively rigid, and do not offer interactive performance. In contrast, our method for computing consistent layouts can adapt to the problem at hand and is inspired from move making algorithms that have been used for image labeling problems.

Discrete Optimization for Image Labeling In computer vision, many tasks such as segmentation of an image can be formulated as image labeling problems where each variable (pixel) needs to be assigned the label which leads to the most probable (or lowest cost/energy) joint labeling of the image. The models for these problems are usually specified as factor-graphs in which the factor nodes represent the energy potential functions that operate on the variables [Kschischang et al. 2001]. In most vision models, the energy function is composed of unary and binary terms and the interactions between objects are generally limited to variables in a 4 or 8 neighborhood grid.

The sparse grid-like structure of the object interactions and the limited number of labels allows for fast solution of image labeling problems using techniques such as graph-cuts [Boykov et al. 2001; Gould et al. 2009; Lempitsky et al. 2010; Szeliski et al. 2006; Woodford et al. 2008], belief-propagation [Pearl 1982], and tree message-passing [Wainwright et al. 2005; Kolmogorov 2006]. In our case rules can be defined over multiple variables, and create complex factor graphs which these approaches do not handle well. Further, each object typically has a large space of possible configurations, which increases the complexity in multi-object interactions. Furthermore, in all but the simplest scenarios the factor graph contains cycles that makes the problem NP-hard even if the label space for each object is small. Our method, however, can deal with such complex factors because of its ability to generate compact encodings of higher order relations by intelligently exploring the space of plausible object placements.

3 Layout Design for Augmented Reality Applications

Using FLARE, a designer specifies a rule-set using declarative programming. First objects are defined, each identified by a unique name and belonging to one of several predefined classes. An object's class defines its properties which may be initialized to a specific value. For each property we can also define a range of acceptable values or define it as fixed (not allowed to change in the optimization process). For example an object might have geometric properties such as position, facing (rotation) and scale, material properties (color, specularity), or physical properties (for physical simulations). For script examples please see the supplemental material.

Rules are written using simple algebraic notation and a library of predefined routines, either as cost functions or as Boolean conditions (in which case we automatically assign a cost function). A rule can reference the properties of any of the objects defined, as well as the environment. For example when arranging objects in a room the designer might reference the type of surface on which an object should be placed (horizontal, vertical). The number of objects included in a rule classify it as unary (one object), binary (two objects) or multiple. We call the space of all possible assignments to the object's properties, the *layout solution space*. We define a cost function

$$cost(s) := \sum_i r_i(\hat{s}_i) \qquad (1)$$

where $r_i : (O_i \subseteq O) \to \mathbb{R}$ is a cost function (rule) operating on a subset of the objects, $s \in S$ is a specific solution, and $\hat{s}_i$ is a slice of the solution containing only the objects in O_i. Typically each rule applies only to a small subset of the objects. An optimal layout for an AR application is one which minimizes the overall cost of its rules.

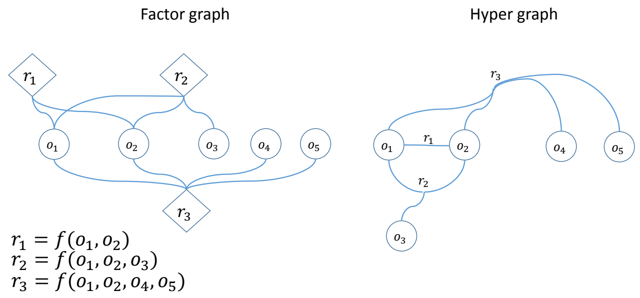

Figure 2: *A design (in this case consisting of three rules over five objects) can be represented as a Factor Graph, a bipartite representation connecting object nodes to factor nodes. Alternatively it can be represented as a hyper-graph in which each node is an object, and an edge represents common rules between connected nodes.*

3.1 Graph Representation

A common graph representation for MAP [1] problems is the Factor Graph [Yeh et al. 2012] which has two node groups: object nodes and factor (rule) nodes. Edges connect factors to the objects they reference (Figure 2 left). A factor representing a unary rule will have one edge, a binary rule will have two edges and so on. We represent a design as a graph $G = (\mathcal{V}, \mathcal{E})$. Each object o has an associated node v_o whose cost function is $\phi(v_o) = \sum\{r|r : \{o\} \to \mathbb{R}\}$ (sum of all unary rules on object o). We connect an edge e between v_{o_1} and v_{o_2} if there exists at least one rule associated with these objects. Its cost function is $\psi_e = \sum\{r|r : \{o_1, o_2\} \to \mathbb{R}\}$. Given an assignment to all of the objects, the summed cost over the nodes and edges of the graph is equal to the design cost (equation 1). Note that a rule may refer to more than two objects, and therefore an edge can connect more than two nodes, creating a hyper-graph (Figure 2 right).

[1] maximum a posteriori probability

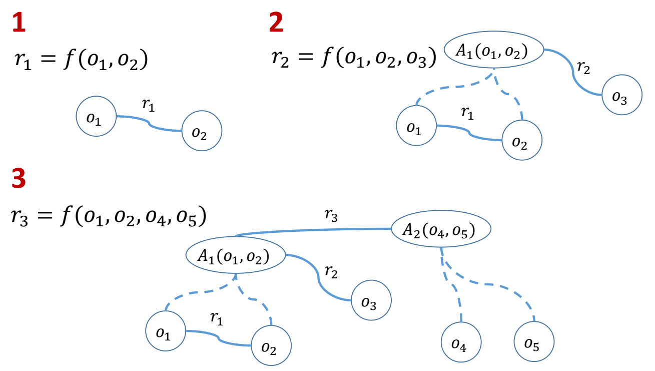

Figure 3: *We construct a graph for a design incrementally. Rules which reference more than two objects are transformed into pairwise interactions via auxiliary nodes: r_1 is a binary rule, r_2 is a ternary rule triggering the creation of A_1 which represents a pair of values (for o_1 and o_2). r_3 involves four variables, in which case we require two auxiliary nodes (we reuse previously created A_1).*

4 Application Layout

An optimal layout is the global minimum in the scalar field defined by the design cost function. Finding the optimal layout or even a good one is difficult: Rule cost functions may be non-convex, rules might be unsatisfiable, for example if they conflict with the environment or with themselves, therefore we cannot know the lower bound on the cost and it is difficult to specify a stopping criteria. And finally, the high-dimensional nature of the space and the assumed sparsity of feasible solutions reduce the effectiveness of stochastic sampling.

Similar to [Merrell et al. 2011; Yu et al. 2011] we focus on a discretized version of the solution space. Given N objects in the design and k possible assignments per object, the size of the solution space k^N makes performing an exhaustive search prohibitively expensive. Previous methods have attempted to sample from the underlying probability distribution function, using Metropolis-Hastings [Hastings 1970] algorithm coupled with concepts from simulated annealing. These methods still require a prohibitively large number of samples (and of course evaluations of the cost function), therefore requiring a long run time or reliance on massively parallel GPU implementations [Merrell et al. 2011]. In many applications performance is an issue, and in some platforms such as mobile devices, computing is costly. Our approach therefore focused on reducing the number of evaluations required to find a feasible solution.

4.1 Transforming High-order rules into Pairwise Interactions

To simplify the graphical representation of the design, we transform hyper-edges into pairwise graph interactions by introducing auxiliary nodes. We divide the set of objects associated with any hyper-edge e into two groups A_1 and A_2. For each group consisting of more than one object, we add an auxiliary node that represents the variables i.e. the value assigned to the auxiliary node encodes the value assigned to all objects represented by this node. If the group contains only one node, then we just use the original object node.

The auxiliary node can only takes values in the space of assignments that satisfy the rules that operate on the group of variables. Therefore, there is zero cost for assigning a particular feasible value to the auxiliary node i.e. $\phi(A_i) = 0$ (no unary cost). We connect the two group nodes with an edge $\hat{e}$ such that $\psi_{\hat{e}} = \psi_e$. We then connect auxiliary nodes with their associated object nodes. The cost function for these edges $\psi(\{o, A_i\})$ is 0 if the assignment to object o matches the assignment to A_i and arbitrarily high otherwise. The addition of the auxiliary variables ensures that there are only binary interactions between nodes. Formally, this corresponds to a cost function:

$$E(x) = \sum_{i \in \mathcal{V}} \phi_i(x_i) + \sum_{ij \in \mathcal{E}} \psi_{ij}(x_i, x_j) \tag{2}$$

where $\mathcal{V}$ and $\mathcal{E}$ represents the set of nodes and the set of edges between these nodes respectively, x_i represents the label taken by a particular node, and ϕ_i and ψ_{ij} are functions that encode unary and pairwise costs. In the second graph of figure 3 we demonstrate how rule r_2 that operates on o_1, o_2 and o_3 is represented by introducing the auxiliary node A_1. The function is then associated with the edge between group nodes (A_1, o_3). Adding r_3 reuses A_1 while adding an additional node A_2 and connecting them.

4.2 Adaptive Layout Space Exploration

A simple method to find a low-cost solution under the function defined in equation 2 is to explore the solution space by local search *i.e.* start from an initial solution and proceed by making a series of changes which lead to solutions having lower energy. At each step, this *move-making* [Lempitsky et al. 2010] algorithm explores the neighboring solutions and chooses the move which leads to the solution having the lowest energy. The algorithm is said to converge when no lower energy solution can be found. An example of this approach is the Iterated Conditional Modes (ICM) algorithm [Besag 1986] that at each iteration optimizes the value of a single variable keeping all other variables fixed. However, this approach is highly inefficient due to the large label space of each variable. Instead we could perform a random walk algorithm, in each iteration we select a new value for one of the objects and evaluate the cost function. Accepting the new configuration with a high probability if the cost improves.

Generating proposals for this algorithm is key to its performance. The most straight-forward approach is to sample uniformly over the object properties. Another approach is to start with uniform sampling (large steps) and over time reduce step size, sampling normally around the previous object value. One such algorithm based on simulated annealing is parallel tempering [Merrell et al. 2011; Yu et al. 2011; Yeh et al. 2012], whose effectiveness relies on a highly-parallel GPU setup. In this approach locally sampled moves are interspersed with switching values between objects, and optimizing in parallel multiple solutions. In scenarios where objects might have a large number of properties (high dimensionality of layout space), objects might be of different classes (different properties) and a highly parallel GPU might not be available, these methods do not fare as well (section 5).

4.3 Exponential-sized Search Neighborhoods

Using bigger moves (sampling in a larger neighborhood) increases the chance of the local search algorithm to reach a good solution. This observation has been formalized by [Jung et al. 2009] who give bounds on the error of a particular move-making algorithm as the size of the search space increases. [Boykov et al. 2001] showed that for many classes of energy functions, graph cuts allow the computation of the optimal move in a move space whose size is exponential in the number of variables in the original function minimization problem. These move making algorithms have been used to find solutions which are strong local minima of the energy (as shown in [Boykov et al. 2001; Komodakis and Tziritas 2005; Kohli et al. 2007; Szeliski et al. 2006; Veksler 2007]).

While traditional move making methods only deal with variables with small label sets, their use has recently been extended to minimizing functions defined over large or continuous labels spaces [Woodford et al. 2008; Gould et al. 2009]. An example of this work is the Fusion move method [Lempitsky et al. 2010] that in principle allows for the minimization of functions defined over continuous variables. The fusion-move algorithm starts from an initial labeling of all the variables. In every iteration of the algorithm, the algorithm proposes a new labeling for all variables. It then chooses for each variable whether to retain its previous label or take the new proposed label. This binary choice problem is solved for all variables simultaneously using graph cuts.

Our method generalizes the above-mentioned algorithms as, in each iteration, instead of proposing a single new labeling for each variable, it proposes multiple new proposals for each variable. [Veksler 2007] had earlier presented a related range-move algorithm in which particular range of labels could be proposed in each iterations. They had used this for problems like single channel image denoising where ranges of intensity values were proposed for every pixel. However, our method differs from this scheme in two specific ways. First, instead of proposing particular ranges of labels, our method proposes arbitrary set of labels for every variables that are carefully selected such that they satisfy all the rules that apply on them. Secondly, our method adaptively selects the number of variables included in the move. In this way it can smoothly explore the whole spectrum of choices between iterated conditional modes on one end (where only one variable is selected), and the full multi-proposal fusion move, that involves changing the label of all variables.

Solving a single iteration We formulate the problem of jointly selecting the best proposals for all variables that satisfy the most rules as a discrete optimization problem. More formally, let $P_i = \{p_i^1, p_i^2, ..., p_i^k\}$ be a set of k proposal configurations for variable x_i. We introduce indicator variables $t_i^l, \forall i \in \mathcal{V}, \forall l \in \{1...k\}$ where $t_i^l = 1$ indicates that variable x_i takes the properties in proposal l. Similarly, we introduce binary indicator variables $t_{ij}^{lr}, \forall ij \in \mathcal{E}, \forall l, r \in \{1...k\}$ where $t_{ij}^{lr} = 1$ indicates that variables x_i and x_j take the position proposed in proposal l and r respectively. Given the above notation, the best assignment can be computed by solving the following optimization problem:

$$\begin{aligned} \min & \sum_{i \in \mathcal{V}} \sum_{l} t_i^l \phi_i(p_i^l) + \sum_{ij \in \mathcal{E}} \sum_{l,r} t_{ij}^{lr} \psi_{ij}(p_i^l, p_j^r) \\ s.t. \quad \forall i, & \qquad \sum_{l} t_i^l = 1 \\ \forall i, j, l, & \qquad \sum_{r} t_{ij}^{lr} = t_i^l \\ \forall i, j, l, r & \quad t_i^l, t_{ij}^{lr} \in \{0, 1\} \end{aligned} \tag{3}$$

The above optimization problem in itself is NP-hard to solve in general. Instead, we solve its LP-relaxation and round the fractional solution. For this purpose, we could use general purpose linear programming solvers. However, we used an implementation of the sequential tree re-weighted message passing algorithm (TRW-S) [Wainwright et al. 2005; Kolmogorov 2006] that tries to efficiently solve the linear program by exploiting the sparse nature of the interactions between variables. TRW-S guarantees a non-decreasing lower bound on the energy, however it makes no assurances regarding the solution (See [Szeliski et al. 2006] for detailed comparisons).

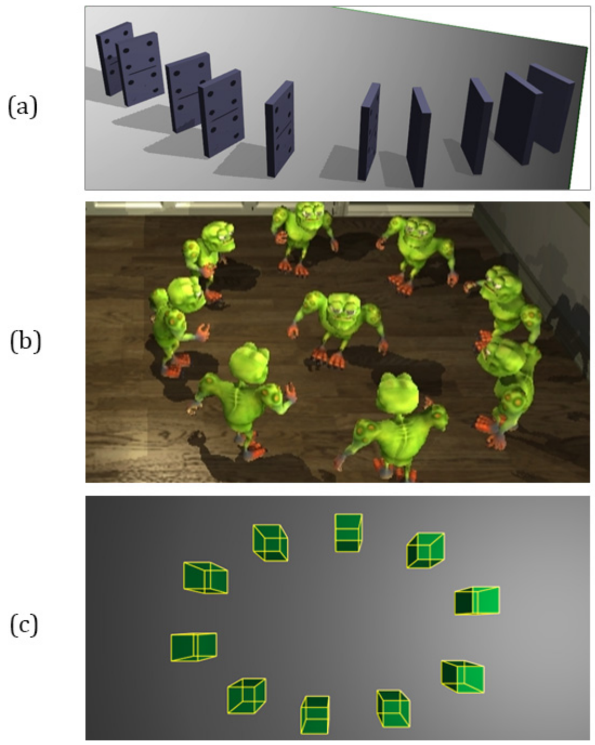

Figure 4: *Visualization of the three qualitative evaluation scenarios: (a) A set of domino tiles set on a curve. Each domino tile is within a set distance from the next, faces in the same direction and approximates a straight line (b) A set of objects arranged in a fixed radius circle around a center object (c) Ten objects such that each one attempts to approximate the average position of both its neighbors, and minimize the distance to them.*

Therefore our revised algorithm works as follows (algorithm 1): Given a design we construct a graph as described in subsection 4.1. **In each iteration** we generate a set of candidates for all objects to be optimized ($ObjectsToOptimize$). In designs with many objects (over 20), we optimize a different random subset of objects in each iteration to reduce the complexity of the graph. In $ProposeCandidates$ we use random sampling and locally-satisfiable proposals (section 4.4) in equal proportions to generate k candidates for each active object.

We evaluate the cost of each rule, for the tuples of values associated with it. Therefore a unary rule is evaluated k times, a binary rule k^2 times and so on. These costs are transferred to the graph nodes and edges as described above. Note that for complex rules, this creates a challenging number of evaluations, which can go up to k^n (where n is the number of objects in the design). We found that by limiting the set of candidates for auxiliary nodes to O(k) tuple values did not reduce the efficiency of the algorithm, and kept our complexity at $O(nk^2)$.

We then attempt to find an improved assignment for our objects, based on the populated graph, using TRW-S. In each iteration, given that we accept the new solution (based on its cost and temperature of the system), we save the new solution and further reduce the temperature (which also reduces the radius of the sampling radius). We repeat for a fixed number of iterations, or until the current accepted solution is beneath a minimum cost.

Algorithm 1 Large Moves

```
procedure LargeMoves($O, R$)                                  ▷ Objects, Rules
    $G \leftarrow ConstructGraph(O, R)$
    $minSolution \leftarrow RandomAssignment(O)$
    $minCost \leftarrow Evaluate(minSolution)$
    for $j \leftarrow 1, niters$ do
        $A \leftarrow ObjectsToOptimize(G, currentSolution)$
        for all $o_i \in A$ do
            $P_i \leftarrow ProposeCandidates(o_i)$
        end for
        for all $r \in R$ do
            UpdateGraphCosts(G, $Evaluate(r, \{P_1, P_2, ...\})$)
        end for
        $currentSolution \leftarrow TRWS(G)$
        $cost \leftarrow Evaluate(currentSolution)$
        if $Accept(cost, minCost)$ then
            $minSolution \leftarrow currentSolution$
            $minCost \leftarrow cost$
        end if
    end for
end procedure
```

4.4 Locally Satisfiable Proposals

The space of possible values (e.g. position, color) of an object is very large and it may require an extremely large number of proposals to obtain a good assignment [Ishikawa 2009]. We overcome this problem by guiding the mechanism through which new proposals are generated. For many types of rules, assignments that satisfy these rules can be found efficiently. In other words, these rules are *locally* satisfiable. In simple terms, given an assignment to some of the objects referenced by r we can generate good proposals for the rest, without resorting to blind sampling in the layout solution space. Our approach could be seen as performing Gibbs sampling [Casella and George 1992], taking advantage of a known partial probability function, to sample from the whole solution space. A few examples follow

1. $dist(a, b) < 4$ is locally satisfiable as given a we generate proposals for b within the circle centered around a with radius 4
2. $collinear(x_1, ..x_n)$ is locally satisfiable given assignments to two of the objects. As we can sample the rest of the objects on the line defined between them.
3. $withinFrustum(a)$ requires a to be in the camera frustum. This is locally satisfiable as generate proposals only from a slice of the 3D space.
4. A constraint on the material properties of two objects, $complementary(a, b)$, is locally satisfiable as given the color of a, the color of b is easy to calculate.

When a designer sets rules in our declarative language, he can define a rule as locally satisfiable. Each such rule has an ”inverse” function which generates proposals for the rule referenced objects, given one or more object assignments. We have found that many designs contain relatively simple geometric constraints, which are very often locally satisfiable.

A *locally satisfiable proposal* (LSP) is a candidate for object o which was proposed by a locally satisfiable rule r. We generate LSP using a greedy strategy. Given the hyper-graph structure of the design, we apply a BFS starting from a randomly selected node. As we discover new nodes, we generate LSP for them, based on the nodes already visited, and the edges by which we discover these nodes. For example consider the following rules applied to objects o_1, o_2, o_3, o_4

$$dist(o_1, o_2) = dist(o_1, o_3) = dist(o_1, o_4) > 1$$

in essence a circle of some radius around o_1. A greedy LSP generation might proceed by selecting o_4 and randomly sampling a position for it. Then we choose the rule $dist(o_1, o_4) > 1$ and select a candidate for o_1 at a distance of at least 1 from o_4. Now given positions for o_1 and o_4, candidates for o_2 and o_3 are generated on the imaginary circle of radius $dist(o_1, o_4)$. Repeating this algorithm creates a series of greedy assignments, which we intersperse with normal sampling, to produce the full candidate set which we evaluate.

5 Experimental Evaluation

We attempt to evaluate both the strength of our rule-based design framework, as well as the benefits of using locally satisfiable proposals to guide our move making algorithm. In section 5.1 we evaluate three constraint sets (design problems), comparing the performance of our algorithm to the prevalent standard of parallel tempering. In section 5.2 we demonstrate three very different AR apps defined using our system, and apply our method to the problem of furniture arrangement in comparison to [Merrell et al. 2011]. In the supplemental material with this submission we show a combination of 2D design layout and photometric mapping application implemented using our system, as further evidence to its wide applicability.

5.1 Quantitative Evaluation

We measure the performance and quality of a layout optimization algorithm by counting rule evaluations. For example calculating the cost of a specific layout for a design is $|r_i|$, the number of rules in the design. Previous approaches have counted the number of samples the algorithm performs for all objects in all iterations. However, this measure favors algorithm which perform an exhaustive search over limited combinations of values. Another measure consists of counting number of evaluations of complete layouts. However, this is not representative of belief-propagation algorithms (such as TRW-S) in which partial evaluations are combined together.

Designs differ in the type and number of rules they contain, and by how constrained the solution is. These differences are reflected in the underlying graph structure, and in our ability to create locally satisfiable proposals. We performed evaluation of our exponential move-making algorithm on three designs, with very different graph structures. For each design we compared the cost of the solution vs. number of evaluations, in comparison to a parallel tempering algorithm we simulated on the CPU. In all three designs the rules are geometric, and each objects in the design can be assigned position, rotation and scale in 2D. For each experiment we ran each algorithm 30 times and took the median of the results.

Domino - Thirty tiles arranged in a curve i.e. each tile t_i has the following rules applied (i) $2 < dist(t_i, t_{i+1}) < 5$ (ii) $\langle t_{i+1} - t_i, t_{i+1}.facing\rangle \leq 0.97$ (iii) $\langle t_{i+1}.facing, t_i.facing\rangle \leq 0.9$. A sample layout can be seen in figure 4(a). The graph is a chain structure which is optimally solved by belief propagation algorithms such as TRW-S. Moreover, LSP is very successful working on non-cyclic graphs as can be seen in figure 5(a).

Circle - In order to test a highly connected graph with cycles, we created a design for nine objects arranged in a circle (with non-fixed radius) around a central object. The minimal angle between any two objects is at least 25^o (example in figure 4(b)). The experiment results are in figure 5(b). All rules in this design are ternary, and the rules enforcing a minimal angle between all objects create a graph with high connectivity. Our algorithm manages to produce good LSP and shows an $x2$ factor over parallel tempering until nearly $120K$ evaluations.

Laplacian Cycle - Finally, to challenge the LSP process, we attempt to arrange ten objects $t_1..t_{10}$ such that $t_i = (t_{i-1} + t_{i+1})/2$ and $d(t_i, t_{i+1}) > C$. Since the rules wrap around t_{10} the cost can never be 0 and the best possible solution is a least-squares oval structure (example in figure 4(c)). In this scenario, where local proposals will never lead to a least-squares solution it is evident that we have no benefit over parallel tempering. Still, as in every iteration our algorithm also performs some random moves, its performance is comparable (figure 5(c)).

5.2 Qualitative Evaluation

Example Apps In order to demonstrate our design framework and layout algorithm we developed several AR apps and games in the Unity 3D game engine [Unity3D 2013]. The environments in which we layout the apps are real rooms captured using a Kinect camera and processed using Kinect Fusion [Newcombe et al. 2011] to extract scene geometry. We process the scene geometry to find a set of surfaces in the room and label them as vertical or horizontal. We implemented our algorithm in a single threaded C# program.

Angry cannon is a physics-based puzzle game in which a user aims a cannon C at brick castles and bomb pillars $b_1, ...b_n$, attempting to knock them down. The castles and pillars are place around the room, within range of the cannon, using existing room features as obstacles. The game objects and their properties are the *cannon* (position, rotation), *brick castles* (position, rotation, number of bricks) and *bomb pillars* (position, rotation, height). The rules are

1. $dist(C, b_i) > 4$
2. $horizontal(C)$
3. $horizontal(b_i)$
4. $collision(C)$
5. $collision(b_i)$

AR Racing is a racing game where the race track is dynamically created for each new room the player visits. Given a desired track length, we create a set of keypoint objects (whose only property is position) $K = \{k_1, ...k_n\}$. The rules for each object k_i are

1. $dist(k_i, k_{i+1}) \in [0.5, 1]$
2. $lineOfSight(k_i, k_{i+1})$
3. $collision(k_i, K/k_i)$
4. $horizontal(k_i)$
5. $collision(k_i)$

where $k_{n+1} \equiv k_1$ and distances are specified in feet. As the tracks grow longer, the keypoints must select different horizontal surfaces in order to preserve the minimal distance, creating complex tracks, taking advantage of the geometry. In order to render the looped track we pass a spline through the keypoints, and on it we place the racing cars.

The *Media Library* application lets a user browse his collection of videos, in any environment. A selection of movies from a database is divided into categories, and displayed on several tile poster objects $p_1, ...p_n$. Each poster has position and facing. Additionally we place two video screens V_1, V_2, meant to hang on the room walls, whose position and scale can change. The rules in this application are

1. $horizontal(p_i)$
2. $vertical(B_i)$
3. $inFOV(p_i)$
4. $inFOV(V_i)$
5. $inner(p_i.facing, eye) \leq -0.8$
6. $collision(p_i)$
7. $collision(B_i)$

Sample results for all three applications can be seen in figures 6,7 and the accompanying video.

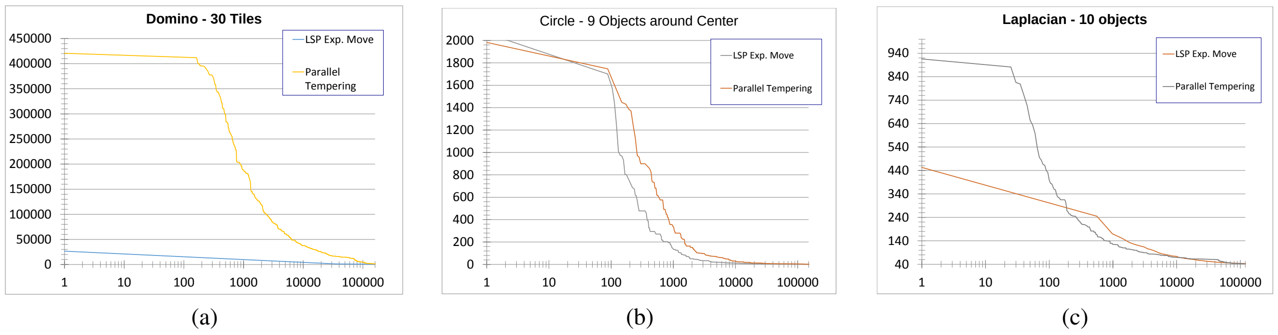

Figure 5: *Experimental results comparing exponential move LSP to parallel tempering performance. In all three graphs the x axis is log-scale number of candidate evaluations and y axis is the solution cost. (a)* Domino*: The chain-like structure of the graph works optimally for our approach. (b)* Circle*: A highly connected graph makes it difficult to converge, but still the LSP approach shows much quicker convergence. (c)* Laplacian Cycle*: The optimal solution is a least-squares one and mostly our approach degrades to random sampling in this example.*

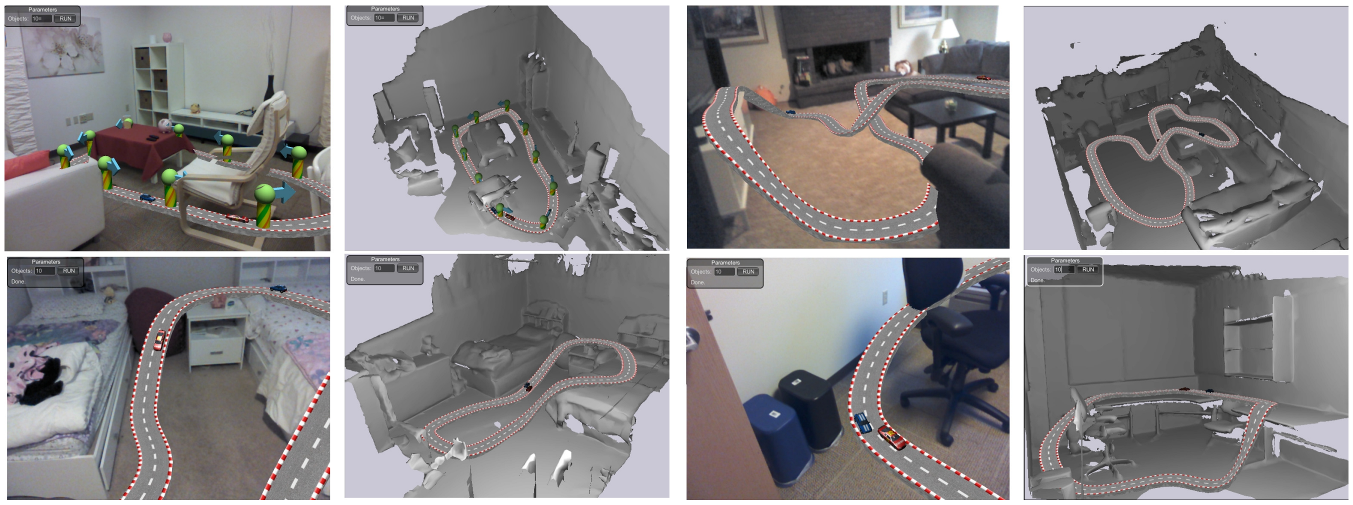

Figure 6: *Results from our racing game application, frame pairs showing the AR rendering of the track mid-game, together with the model of the room.*

Furniture Arrangement We recreated the design rules described in [Merrell et al. 2011], rewriting them to be locally satisfiable. We then used our algorithm to find furniture layout in several room configurations (figure 8). Our approach produced comparable results in 50000 evaluations, compared to $5M - 10M$ evaluations (extrapolated from figure 7 in their paper). Note that while we produce a single feasible solution in each run, they attempt to produce a variety of solutions.

6 Discussion and Future Work

We presented FLARE, a rule-based design framework for AR applications. A designer defines application components as objects with layout properties, and a set of rules which help target the application to any environment. The environment is represented by a set of features is extracted from recovered geometry and the color video taken at the scene.

The richness of the rules is partially dependent on the features extracted from the scene. In this paper, we demonstrated using planar features as they are common in indoor scenes and were sufficient to generate all the examples in the paper. Other environments, such as natural scenes, may require other features. In the supplemental material we adapt our framework for 2D design, employing saliency and color features, as a demonstration of the flexibility of the system.

At the core of our system is a novel algorithm for rule-based design layout problems. Our approach unifies and expands on previously proposed local search based methods. We introduced the concept of locally satisfiable proposals and demonstrated that their use dramatically reduces the number of evaluations required for finding a rule-consistent layout. In cases where LSP fails, our algorithm degrades to a random sampling approach.

All the examples shown in this paper were generated automatically, from the geometry reconstruction, to the plane extraction and targeting the different apps to the environment. However, the mapping is not without limitations. It is possible to assign a set of rules that will not be satisfied in a given environment. For example, we might wish for an object to be positioned on an elevated horizontal surface above the floor, which may not exist in a given room. In this case the optimal cost function for the design cannot be 0 and the system will approach that minimum (e.g. place the object on the floor). In designs where the optimal solution would be a least-squares solution, our locally-satisfiable proposals do not provide a benefit and our algorithm degrades to random sampling.

In the future we hope to port our algorithm to a massively parallel GPU implementation similar to parallel tempering, and adapt it to other design problems. We have a strong belief that immersive augmented reality will see a surge in research over the next few

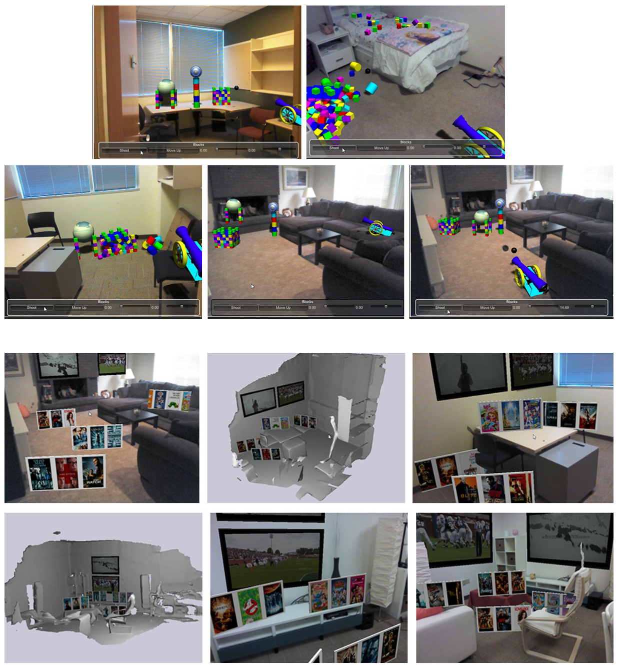

Figure 7: *(top) Results from our angry cannon game. (bottom) Results from our media library application.*

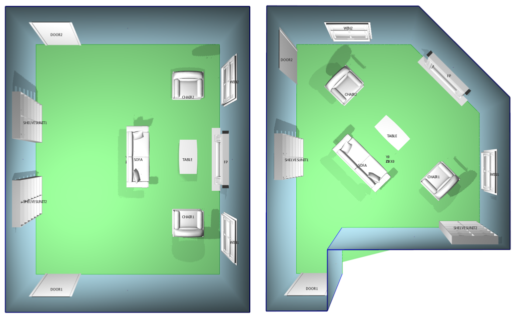

Figure 8: *We follow the design rules described in [Merrell et al. 2011] and apply our move-making algorithm to generate these furniture arrangements.*

years and hope our system can serve as a basis for other mapping algorithms.

References

ARCHITIP, 2013. Website. `http://architip.mobi`.

AZUMA, R., BAILLOT, Y., BEHRINGER, R., FEINER, S., JULIER, S., AND MACINTYRE, B. 2001. Recent advances in augmented reality. *IEEE Computer Graphics and Applications 21*, 6 (Nov.), 34–47.

BESAG, J. 1986. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society. Series B (Methodological)*, 259–302.

BORNING, A., AND FREEMAN-BENSON, B. 1998. Ultraviolet: A constraint satisfaction algorithm for interactive graphics. *Constraints 3*, 1, 9–32.

BOYKOV, Y., VEKSLER, O., AND ZABIH, R. 2001. Fast approximate energy minimization via graph cuts. *PAMI 2001*.

CARMIGNIANI, J., FURHT, B., ANISETTI, M., CERAVOLO, P., DAMIANI, E., AND IVKOVIC, M. 2011. Augmented reality technologies, systems and applications. *Multimedia Tools and Applications 51*, 1 (Jan.), 341–377.

CASELLA, G., AND GEORGE, E. I. 1992. Explaining the gibbs sampler. *The American Statistician 46*, 3, 167–174.

FISHER, M., RITCHIE, D., SAVVA, M., FUNKHOUSER, T., AND HANRAHAN, P. 2012. Example-based synthesis of 3d object arrangements. In *ACM SIGGRAPH Asia 2012 papers*, SIGGRAPH Asia '12.

GOULD, S., AMAT, F., AND KOLLER, D. 2009. Alphabet soup: A framework for approximate energy minimization. In *CVPR 2009*, 903–910.

GRASSET, R., GASCUEL, J.-D., AND SCHMALSTIEG, D. 2003. Interactive mediated reality. In *ISMAR 2003*.

HASTINGS, W. K. 1970. Monte carlo sampling methods using markov chains and their applications. *Biometrika 57*, 1, 97–109.

ISHIKAWA, H. 2009. Higher-order gradient descent by fusion-move graph cut. In *ICCV 2009*, 568–574.

JONES, B. R., BENKO, H., OFEK, E., AND WILSON, A. D. 2013. Illumiroom: Peripheral projected illusions for interactive experiences. In *CHI 2013*.

JUNAYO, 2013. Website. `http://www.junaio.com`.

JUNG, K., KOHLI, P., AND SHAH, D. 2009. Local rules for global map: When do they work ? In *NIPS*, 871–879.

KOHLI, P., KUMAR, M. P., AND TORR, P. H. S. 2007. P3 & beyond: Solving energies with higher order cliques. In *CVPR*.

KOLMOGOROV, V. 2006. Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. Pattern Anal. Mach. Intell. 28*, 10 (Oct.), 1568–1583.

KOMODAKIS, N., AND TZIRITAS, G. 2005. A new framework for approximate labeling via graph cuts. In *ICCV*.

KSCHISCHANG, F. R., FREY, B. J., AND LOELIGER, H.-A. 2001. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory 47*, 2, 498–519.

KUMAR, V. 1992. Algorithms for constraint satisfaction problems: A survey. *AI MAGAZINE 13*, 1, 32–44.

LAYAR, 2013. Website. `http://www.layar.com`.

LEMPITSKY, V. S., ROTHER, C., ROTH, S., AND BLAKE, A. 2010. Fusion moves for markov random field optimization. *IEEE Trans. Pattern Anal. Mach. Intell. 32*, 8, 1392–1405.

LIN, S., RITCHIE, D., FISHER, M., AND HANRAHAN, P. 2013. Probabilistic color-by-numbers: Suggesting pattern colorizations using factor graphs. In *ACM SIGGRAPH 2013 papers*, SIGGRAPH '13.

MACKWORTH, A. K. 1977. Consistency in networks of relations. *Artificial Intelligence 8*, 1, 99 – 118.

MARGRIET SCHAVEMAKER, HEIN WILS, P. S., AND PONDAAG, E. 2011. Augmented reality and the museum experience. In *Museums and the Web 2011*.

MERRELL, P., SCHKUFZA, E., LI, Z., AGRAWALA, M., AND KOLTUN, V. 2011. Interactive furniture layout using interior design guidelines. In *SIGGRAPH 2011*.

NEWCOMBE, R. A., IZADI, S., HILLIGES, O., MOLYNEAUX, D., KIM, D., DAVISON, A. J., KOHLI, P., SHOTTON, J., HODGES, S., AND FITZGIBBON, A. W. 2011. Kinectfusion: Real-time dense surface mapping and tracking. In *ISMAR*, IEEE, 127–136.

ONG, S. K., AND (EDS.), A. Y. C. N., Eds. 2004. *Virtual and Augmented Reality Applications in Manufacturing*.

PEARL, J. 1982. Reverend bayes on inference engines: A distributed hierarchical approach. In *AAAI*, 133–136.

SZELISKI, R., ZABIH, R., SCHARSTEIN, D., VEKSLER, O., KOLMOGOROV, V., AGARWALA, A., TAPPEN, M. F., AND ROTHER, C. 2006. A comparative study of energy minimization methods for markov random fields. In *ECCV 2006*.

UNITY3D, 2013. Unity. Website. `http://unity3d.com/`.

VEKSLER, O. 2007. Graph cut based optimization for mrfs with truncated convex priors. In *CVPR 2007*.

WAINWRIGHT, M. J., JAAKKOLA, T., AND WILLSKY, A. S. 2005. Map estimation via agreement on trees: message-passing and linear programming. *IEEE Transactions on Information Theory 51*, 11, 3697–3717.

WIKITUDE, 2013. Website. `http://www.wikitude.com`.

WOODFORD, O. J., TORR, P. H. S., REID, I. D., AND FITZGIBBON, A. W. 2008. Global stereo reconstruction under second order smoothness priors. In *CVPR*, IEEE Computer Society.

YEH, Y.-T., YANG, L., WATSON, M., GOODMAN, N. D., AND HANRAHAN, P. 2012. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM Trans. Graph. 31*, 4 (July), 56:1–56:11.

YU, L.-F., YEUNG, S.-K., TANG, C.-K., TERZOPOULOS, D., CHAN, T. F., AND OSHER, S. J. 2011. Make it home: automatic optimization of furniture arrangement. In *SIGGRAPH 2011*.