Game Level Layout from Design Specification

Chongyang Ma et al.
EUROGRAPHICS 2014

Presented by YoungBin Kim
2015. 01. 13

Computer Graphics @ Korea University
Abstract

- Propose an algorithmic approach
  - Automatically laying out game levels from user-specified blocks
  - Designers to retain control of the gameplay flow
    - Via user-specified level connectivity graphs
  - Produces sequences of diverse layouts
    - For the same input connectivity
Introduction
Introduction

• Typical video games
  ▪ Contain complex virtual environments, or game levels
  ▪ Players must traverse in order to advance through the story
    • Each level is a series of spaces, or rooms, with connections

• Rich, interesting game levels
  ▪ Key component of a successful game
Introduction

• Level layouts
  ▪ Typically manually constructed by game designers
  ▪ Starting from a desired gameplay flow
    • Construct each level
      ▪ From a set of artist-created building blocks, or spaces
    ▪ Player experience never changes

• Roguelikes
  ▪ Offer randomly generated levels
    • Designer control is lost
      ▪ Can no longer steer the flow of gameplay
Introduction

• Propose a novel approach
  ▪ For game level layout generation
    • Capable of automatically producing
      ▪ Variety of distinct game levels for replay
      ▪ While still allowing the designer to define the flow of gameplay

• Input to our method
  ▪ Planar connectivity graph
    • Reflects the designer-intended gameplay flow
  ▪ Set of polygonal building blocks
Technical challenge
  - Layout the blocks such that two key constraints are satisfied
    - Layout must be *intersection-free*
      - Require that blocks within each level do not overlap
    - Must also satisfy all pairwise *contacts*
      - Each pair of blocks connected by a graph edge
      - Must share a boundary segment long enough
        - Place a doorway through
  - Game level layout a challenging problem
1. Exploit graph connectivity to design
   - Divide-and-conquer layout strategy
     - Speeding up convergence to valid solutions
2. Considers reduced, continuous configuration spaces
   - Defined for pairs of adjacent blocks
     - Instead of exploring the set of all possible building block positions
     - Quickly evaluate contacts
     - Instantaneously improve them for individual blocks

   - These two ingredients
     - Efficiently explore the solution space
Introduction

- Our contribution
  - First game level layout algorithm for general 3D maps
    - Provides high-level designer-control of gameflow
    - While enabling the use of existing game building blocks
  - Easily be integrated into existing game development workflows
    - Suitable for a wide range of genres
  - Practical solution to a challenging layout problem
Background and Related Work
Background and Related Work

- **Industry Practices**
  - [Per02, BP13]
    - Create a set of templates, or building blocks
    - Defining possible space shapes
    - Reuse those multiple times when assembling a level
      - Allows for easy assembly of the final 3d realizations of the layouts

- **Automating Level Generation**
  - Dungeons of Dredmor [Gas11]
    - Handle only acyclical layouts
    - Use blocks quantized
      - Aligned with a fixed grid
Background and Related Work

• Automating Level Generation
  - [JYT10, ALM11]

• Do not support the predefined blocks
Background and Related Work

• 3D architectural shapes
  ▪ [WWSR03, MWH*06, MM08, LCOZ*11]

• 2D building layouts
  ▪ [PM01, CEW*08, VKW*12, YYW*12]

• Plausibility and aesthetics
• Do NOT control contacts or adjacencies
Background and Related Work

- **Floor plans**
  - [MS74, Sha87, Lig00, MSK10, LYAM13, BYMW13]

- **VLSI circuits**
  - [She98]

- **Controlled envelope**
- **Axis-aligned elements only**
Background and Related Work

- 3D Game Levels
  - [YWR09, KW11, MM08, CLDD09]
    - 2D game levels and floorplans can be converted
      - Into 3D architectural structures
    - Our outputs can directly be used as the input to their method
Algorithm
Algorithm

• Technical challenges
  ▪ High-dimensional
    • Large number of blocks
  ▪ Mixed continuous-discrete search space
    • Continuous : block positions
    • Discrete : node-to-block associations
  ▪ Naïve stochastic optimization
    • Low convergence rates
Configuration Space

• Specifically given a pair of blocks
  ▪ One fixed and one free
  ▪ Set of valid positions is a union of 1D line segments
    • Can be computed analytically
  ▪ Robotics literature are referred to as configuration spaces
    • [LP83, LaV06]
  ▪ Dramatically reduce the size of the space that we must search
Configuration Space

- Configuration space of a moving block
  - With respect to two or more stationary ones
    - Intersection of collections of line segments
    - Usually points
    - Can be line segments
      - Block can contact both of its neighbors
        - Along two parallel or collinear edges
Incremental Layout

• To speed up processing
  ▪ Break the layout problem into, smaller, easier to solve ones
    • Each node has at most two neighbors
      ▪ Easy to lay out
  
• Our strategy
  ▪ Decomposing the graph into chains
    • Cyclical chains
      ▪ Significantly more constrained than open-ended ones
  ▪ Deciding on chain processing order
Incremental Layout

• Decomposing the graph into chains
  - Decompose the input graph into a set of cycles and trees
    • Using a planar embedding of the input graph generated
      - Using a standard algorithm
        • [CP89]

A: \{A,C\}, \{A,D\}, \{A,B\}, \{A,E\}
B: \{B,A\}, \{B,D\}, \{B,E\}
C: \{C,A\}, \{C,E\}, \{C,D\}
D: \{D,C\}, \{D,E\}, \{D,B\}, \{D,A\}
E: \{E,A\}, \{E,B\}, \{E,D\}, \{E,C\}

• Then find all the faces in the embedding
Incremental Layout

- Deciding on chain processing order
  - Form the first chain using the edges of the face
    - With the minimum number of edges in the embedding
  - Iteratively consider neighboring planar faces
    - Picking a face
    - Grouping all edges on that face
      - That are not already in a cycle into a new cycle
    - Given multiple neighboring chains
      - Select the shorter ones first
    - Repeat the process until all faces are processed
  - Remaining acyclical graph components
    - Processed in a breadth-first order
Incremental Layout

1. Compute a set of possible layouts for one chain
2. Then extend these partial layouts by adding neighboring chains
3. Generating multiple extended layouts and storing them
   • Process terminates
     ▪ All chains are processed
     ▪ Sufficient number of layouts have been found
     ▪ Or when no more distinct layouts can be computed

**Pseudocode 1 Incremental level layout**

**Input:** Planar graph $G$, building blocks $B$, layout stack $S$

1. **procedure** INCREMENTAL_LAYOUT$(G, B, S)$
2. Push empty layout into $S$
3. **repeat**
4. $s \leftarrow S$.pop()
5. Get the next chain $c$ to add to $s$
6. AddChain$(c, s)$ //extend the layout to contain $c$
7. if extended partial layouts were generated then
8. Push new partial layouts into $S$
9. **end if**
10. until target # of full layouts is generated or $S$ is empty
11. **end procedure**
**Incremental Layout**

- Goal of generating multiple partial layouts
  - Increase the likelihood
    - At least one of these layouts can be extended to a full layout
  - Pre-caching partial layouts
    - Facilitate quick, on-demand, creation of additional full layouts
- If an extension step fails
  - Backtrack to the last previously computed and stored partial layout
  - Continue the extension process from it
Chain Layout

- Goal of the chain layout step
  - Need to find blocks and positions
    - For the nodes of the added chain
      - Such that the extended layout is valid
- To assist the layout of this new chain
  - Allow changes to the input partial layout
    - But keep the probability of those low
Chain Layout

- Simulated annealing framework
  - Built-in randomization process
  - Energy function
    \[ E = e^{\frac{A}{\sigma}} \cdot e^{\frac{D}{\sigma}} - 1 \]
    - \( A = \) Total area of intersection
      - Between two blocks in the layout
    - \( D = \) Sum of squared distances
      - Between the center of pairs of blocks
        - That are connected in the extended subgraph
        - But which are not in contact
    - \( \sigma = \) One hundred times the average block area
      - Smaller value of \( \sigma \) produces a higher success ratio
      - Larger value of \( \sigma \) produces a quicker convergence
      - Reasonable trade-off between the two
Chain Layout

Pseudocode 2: Extend partial layout $s$ adding the chain $c$

```plaintext
procedure AddChain($G, B, S, c, s$)
1: $t \leftarrow t_0$ // Initial temperature
2: for $i \leftarrow 1, n$ do // $n$: # of cycles in total
3: for $j \leftarrow 1, m$ do // $m$: # of trials per cycle
4: $s' \leftarrow$ Locally perturb $s \cup c$
5: if $s'$ is valid then
6: if $s \cup c$ is full layout then output it
7: else if $s'$ passes variability test
8: Push $s'$ into $S$
9: Return if enough extended layouts computed
10: end if
11: end if
12: if $\Delta E < 0$ then // $\Delta E = E(s') - E(s)$
13: $s \leftarrow s'$
14: else if rand() $< e^{-\Delta E/(k*t)}$ then
15: $s \leftarrow s'$
16: else
17: Discard $s'$
18: end if
19: end if
20: end for
21: $t \leftarrow t \times \text{ratio}$ // Cool down temperature
22: end for
23: end procedure
```

- **Parameters**
  - $n = 50$
  - $m = 500$
  - $t_0 = 0.6$
    - Initial temperatures
  - $t_1 = 0.2$
    - Final temperatures
  - $k = \Delta E$ averaging
Chain Layout

• Initialization
  ▪ To speed up processing
    • Aim to start the annealing in a low energy configuration
    • Generate a BFS ordering of the chain blocks starting
      ▪ With those adjacent to the input partial layout if one exists
      ▪ Or with a random root block otherwise
    • Place the blocks one at a time
      ▪ Sampling their configuration space
        • With respect to their already laid out neighbors
      ▪ Selecting the sampled position with the lowest energy
Chain Layout

- **Local Perturbation**
  - Perturb the current layout
    - By changing the block position
    - By changing the node-to-block association
  - When deciding on the node to perturb
    - Only consider nodes in the currently added chain
    - And nodes in the previous layout with non-zero energy
  - Set the ratio
    - Position perturbations to node-to-block association changes
      - 7 to 3
Chain Layout

- **Local Perturbation**
  - Compute the intersection of all configuration spaces of the block
    - With respect to each individual adjacent block
  - Then randomly sample this intersection space
    - To obtain a new position
  - If this intersection is empty
    - Compute the maximal non-empty intersection of spaces
    - And sample it instead
  - Using this approach
    - Locally maximize the number of contacts
      - Pushing the layout closer to a desirable global solution
Chain Layout

- **Processing Valid Layouts**
  - After each local perturbation
    - Check to see if the extended layout is now valid
      - Each block within the configuration space of its neighbors
      - Also check for intersections between pairs of non-adjacent blocks
  - Each valid layout
    - Which differs sufficiently from previously located ones
    - Placed on the stack for further processing
      - By the incremental layout framework
    - Similarity between two partial layouts
      - Defined as the sum of squared distances
        - Between corresponding block centers
        - After centering the two layouts around the origin
Chain Layout

• **Processing Valid Layouts**
  - Terminate the algorithm
    • When enough newly valid extended layouts have been found
  - Partial layouts
    • Generate up to 15 variations
  - Layout of the entire input graph
    • User-specified target number of layouts
Results
Results

- Different building blocks
Results

- Large input graph
Results

- High-valence tree
Results

- Multi-floor constraints
Results

- Boundary constraints
Results

• Restricted door positions
Results

- **Statistics**
  - To evaluate speed and robustness
  - Ran the algorithm 50 times
    - With different randomization seeds
      - For each combination of input planar graph and building blocks
    - Record the rate of successfully computed valid solution
    - As well as the runtime required to find the first solution
    - Also the time and iteration count
      - Necessary to compute ten solutions

<table>
<thead>
<tr>
<th></th>
<th>success rate</th>
<th>#sol.</th>
<th>first solution</th>
<th>ten solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>first/ tenth</td>
<td></td>
<td>time</td>
<td>time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>avg/ med</td>
<td>avg/ med</td>
</tr>
<tr>
<td>Fig 1</td>
<td>1/0.94</td>
<td>9.8</td>
<td>4.9/2.3</td>
<td>10.9/6.8</td>
</tr>
<tr>
<td>Fig 7, top</td>
<td>1/1</td>
<td>10</td>
<td>1.1/0.4</td>
<td>1.8/1.2</td>
</tr>
<tr>
<td>Fig 7,bot</td>
<td>0.94/0.84</td>
<td>9.3</td>
<td>23/18</td>
<td>48/40</td>
</tr>
<tr>
<td>Fig 8</td>
<td>0.98/0.98</td>
<td>10</td>
<td>80/55</td>
<td>94/73</td>
</tr>
<tr>
<td>Fig 9</td>
<td>1/1</td>
<td>10</td>
<td>1.7/0.3</td>
<td>2/0.6</td>
</tr>
</tbody>
</table>
Results

- **Impact of Design Choices**
  - Statistic for algorithmic alternatives measured on graphs

<table>
<thead>
<tr>
<th></th>
<th>success rate first/tenth</th>
<th>#sol.</th>
<th>first solution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>avg/med</td>
</tr>
<tr>
<td>NO config. space</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fig 7, top</td>
<td>0.4/0</td>
<td>1</td>
<td>64/64</td>
</tr>
<tr>
<td>Fig 9</td>
<td>0.7/0</td>
<td>1</td>
<td>51/38</td>
</tr>
<tr>
<td>NO inc. layout:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fig 7, top</td>
<td>0.7/0</td>
<td>3.5</td>
<td>1.4/0.8</td>
</tr>
<tr>
<td>Fig 9</td>
<td>0.76/0</td>
<td>3.8</td>
<td>1.6/1.2</td>
</tr>
</tbody>
</table>
Results

- **Impact of Design Choices**
  - Asked ten professional game developers
    - From independent and AAA studios
    - To review our generated layouts
  - Feedback was uniformly positive
    - "nice and natural"
    - "more complex than can easily be made by hand"
    - "layout and packing feel good"
    - "brainstorming tool for exploring level ideas"
Conclusions
Conclusions

• Conclusion
  ▪ A novel level layout synthesis method
    • For various design goals
  ▪ A *graph-decomposition* based layout strategy
    • For complex connectivity
  ▪ A stochastic optimization algorithm based on *configuration space*
    • For fast convergence

• Future Work
  ▪ Additional design goals
    • Production scenarios
  ▪ Speedup
    • More advanced stochastic search
  ▪ Increase output variability
    • Allow block deformation