Jigsaw Image Mosaics

Kim and Pellacini, 2002
Kacper Wysocki

26th January 2005

Notes

- Given arbitrary container and set of aribitrary tiles, fill compactly with similar color, optionally deforming slightly for effect.
- **Problem:** given container image and tiles $\{T_i\}$, find set of shapes $\{S_j\}$ such that
  - union over S_j resembles container image as closely as possible
  - each S_j is a translated and rotated copy of one of the T_i's, possibly slightly deformed

Energy framework for mosaicing

- Minimize the weighted sum of energy terms. Change weights of terms ==> different results.
- Generalizes *Photomosaics* and *Simulated Decorative Mosaics*
- Maintain edges using best-fitting tiles (ie wedge-shaped in corners)

Advantages

- user can control result by changing weights
- introduce new mosaicing generation rules by adding more terms to energy function
- preparation and generation is completely automatic

Contributions

- energy-based framework for mosaicing generalizing known algorithms
- energy-minimization algo solving mosaicing problem at acceptable cost
- 'soft' packing problem -> feature-based texture synthesis and product manufacturing

Related work

- *Photomosaics* [Finkelstein and Range, 1998] [Silvers and Hawley, 1997]
  - collection of images arranged in rectangular grid
  - for each block, search db for closest match
  - quick and impressive, but limited to rectangular
- *Simulated Decorative Mosaics* [Hausner, 2001]
  - align square tiles with varying orientations to preserve input edges and maximize coverage
  - we use arbitrary shapes, can't use algo directly

We can do the above, but slower.

- *Escherization* [Kaplan and Salesin, 2000]
  - regular tilings, using closed figure, as close as possible to original figure
- [Haeberli, 1990]
  - randomly choose tile positions, construct CVD, fill each voronoi region with sampled color
  - may not fit tiles
- Packing problem is NP-hard
  - boundary matching
  - db-driven layout
  - leftmost placement policy
  - *Dense Packing of Poly's* [Milenkovic, 1999]
    - computational geometry and math programming

Preparing inputs

Input:

- container image
- set of tiles
- shape of tiles and container as polygons

Use *active contours*[Kass et al., 1987]

- automatic segmentation from clip art harvested from web
- important edges: segment input image into disjoint containers - final composite will have important edges
- algorithm independent within each segment
- allow user-specified arbitrary segmentations

Mosaicing Framework

Formalization

- Tile configuration: subset of input tiles with repetition + transformations
- **JIM when minimizes E in**

$$E = w_C \cdot E_C + w_G \cdot E_G + w_O \cdot E_O + w_D \cdot E_D$$

C	color difference
G	gap
O	overlay/penetration
D	deformation

Photomosaics:

- use rectangular tile db
- $w_G = w_O = w_D = \infty$

Simulated Decorative Mosaics:

- square tiles with uniform color chosen from input image palette
- segment container to preserve edges
- $w_D = \infty$

Intuitive use of weights

Energy evaluation

E_C average L^2color differences at random locations, for each tile

E_G spring energy formulation: each vertex attached with spring to nearest edge. if signed distance is > 0, add $E_G = \frac{d^2}{2}$

E_O same as E_G, but for d < 0

E_D sum of deformation energies for each tile - difference in shape from original

$$E_D = \frac{1}{2}\sum_{i=1}^{k}\int_0^1 \alpha|D_i''(s) - T_i''(s)|^2 + \beta|D_i'''(s) - T_i'''(s)|^2 ds$$

where $T_i(s)$ and $D_i(s)$ are original and deformed shapes of the i-th tile, parametrized by $s \in [0, 1]$. First and second term of integral measure the difference wrt stretching and flexing, respectively, while α and β are sensitivity params.

Basic algorithm

Three phases:

1. Place/pack tiles, ignoring deformation
2. Refine and deform
3. Assemble, adjust

works because deformations are always smaller than smallest tile

1. **Packing:**
   - approx by ignoring deformations
   - one tile at a time
   - search db for tile
   - determine exact position & orientation to maximally align against boundry
   - **Registration problem**
   - keep placing tiles until full or cannot place
     - backtrack to previous good energy config
   - new container = old container - shape

2. **Refine**

   - deform to reduce gaps and overlaps
   - compute final deformation using active contours interacting with eachother
     –

$$w_c \cdot \nabla E_C + w_G \cdot \nabla E_G + w_O \nabla E_O + w_D \cdot \nabla E_D = 0$$

     – ∇E_C is close to 0
     – $\nabla E_O = 2d \cdot n$ or gap, shrink/expand
     – $\nabla E_D = \alpha(D_i''(s) - T_i''(s)) + \beta(D_i'''(s) - T_i'''(s))$

Optimizations

Time complexity is

$$O(V_{tile} \cdot N_{tile} \cdot V_{container} \cdot N_{tilesIn} \cdot (1 + b))$$

V_{tile}	number of vertices per tile
N_{tile}	number of tiles in database
$V_{container}$	vertices in container
$N_{tilesIn}$	tiles in container
b	branching overhead

Tile placement

Reduce branch factor b

- try locations that make container easier to fill after update
- guess container after 'average' tile
- easier to fill if as convex as possible
- so:
  – construct CVD with areas roughly size of average tile
  – pick random least-neighboured site

Branch-and-bound with lookahead

Penalize tiles that make filling more difficult at next iteration

Add term to energy eq'n:

$$\begin{aligned} E &= w_G \cdot E_G + w_O \cdot E_O + w_C \cdot E_C + w_{LA} \cdot E_{LA} \\ E_{LA} &= w_A \cdot area + (1 - w_A) \cdot length^2 \end{aligned}$$

favours small area and short circumference, prevents tiles that fit well but lead to hard-to-fill updated container

Container cleanup

after update, jagged or disjoint edges in container

If shallower than shallowest tile, will never be filled. Separate regions and consider gap.

Reduces $V_{container}$ and branching factor

Geometric Hashing

- Match geometric features against database of features
- find a set of suitable tiles, then evaluate energy equation
- pruning technique

Preprocessing

- grid of squares in the plane = table entries
- if shape boundry crosses square, record tile ID and orientation as entry in list attached to table entry
- place all tiles in all discrete orientations in grid to build hash table

Packing stage

- Register container boundry segment to hash table
- access entries of squares container passes through
- for every tile found, cast vote for (tile ID, orientation) pair
- consider entries with more than {treshold} votes
- Reduces $O(N_{tile})$ to $O(h_{grid})$ where h_{grid} is grid granularity

Results

- 900 tiles
- 8x size variations
- 10 min up to 2 hr

Conclusions

- general energy-based framework for mosaicing problems generalizing exisiting algo's
- JIM
- good 'soft' packing for texture synthesis and product manufacturing

Future work

- bounds for energy of final configuration are difficult to predict
- 3D mosaics for surface and volume packing
- video mosaics - Klein et al.