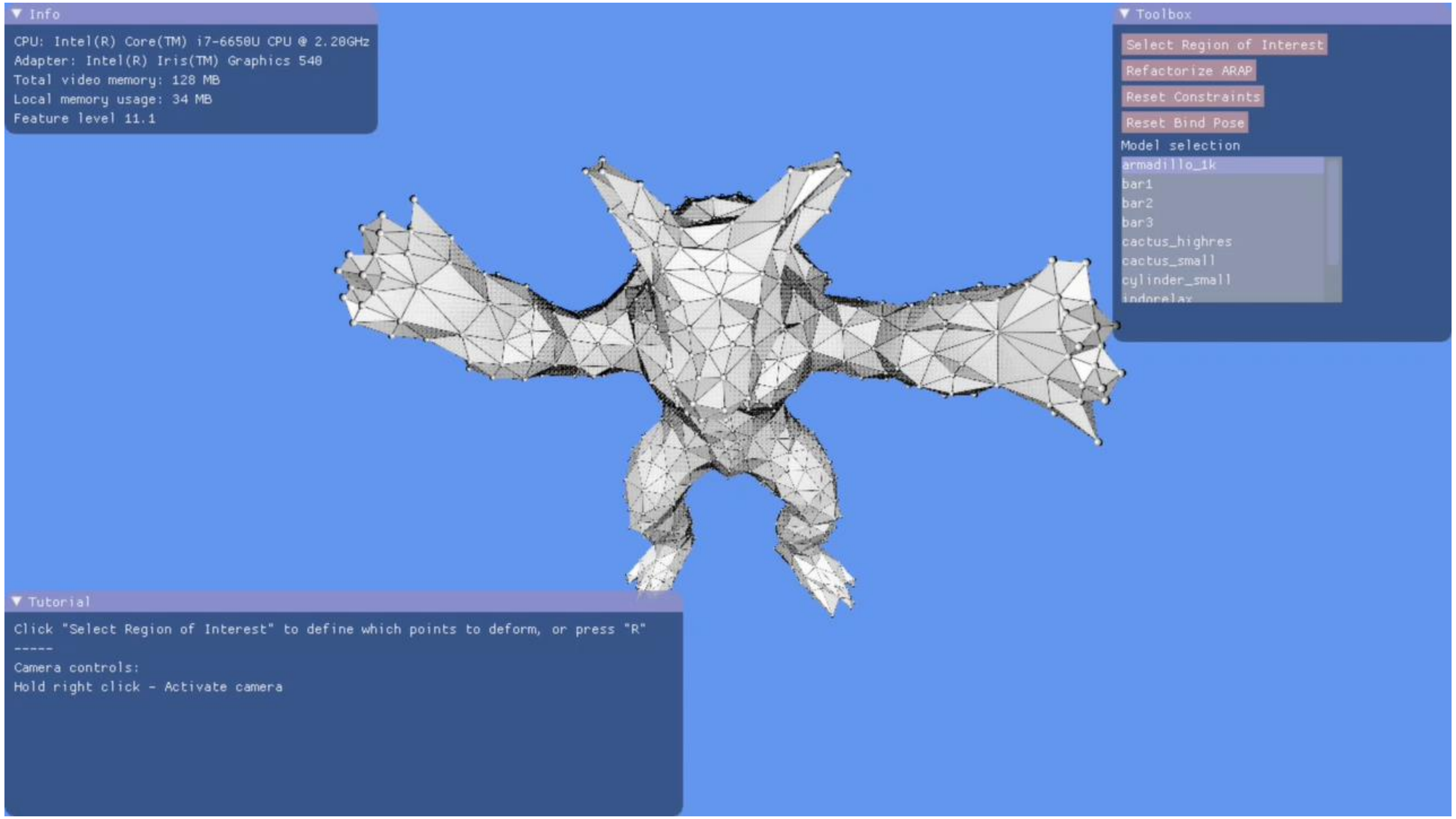
ARAP Implementation Notes

Implementation Video



https://youtu.be/Jl3hluJ8LOo

Required pre-reading

- http://www.igl.ethz.ch/projects/ARAP/arap_web.pdf
- http://doc.cgal.org/4.5/Surface_modeling/index.html

Building instructions

- **exe is supplied, you can just run it.**
- Otherwise, open and build vsproj/glowing-telegram.sln
- Build dependencies:
  - D3D11, Win10 SDK
  - Intel Math Kernel Library
- Hardware requirements:
  - Feature Level 11.0 GPU (Tested: Intel Iris Graphics 540, NVIDIA GTX 970)
  - **AVX2 compatible CPU** (Tested: Intel 6650U, Intel 5960x)
- Software requirements:
  - Windows 10, 64-bit

API (arap.h)

Initializing the system matrix (done every time constraints change):

```
arap_system* create_arap_system_matrix(...¹);
void destroy_arap_system_matrix(arap_system* sys);
```

At every update:

```
void arap(arap_system* sys, ...¹);
```

1: "..." = halfedges, positions, weights, constraints, iterations, etc.

Algorithm

```
void arap(
    arap_system* sys, const float* p_bind_XYZs, float* p_guess_XYZs,
    const int* v_hIDs, const int* h_vfnpIDs, const float* e_ws, int ni)
{
    // iteratively refine guess by optimizing rotation and position
    for (int iter = 0; iter < ni; iter++)
    {
        update_rotations(
            sys, p_bind_XYZs, p_guess_XYZs, v_hIDs, h_vfnpIDs, e_ws);
        update_positions(
            sys, p_bind_XYZs, p_guess_XYZs, v_hIDs, h_vfnpIDs, e_ws);
    }
}
```

Optimizing Rotation

- For each vertex i, compute $S_i = \sum_{j \in N(i)} w_{ij}\, e_{ij} {e'_{ij}}^T$
- Compute SVD $S_i = U_i \Sigma_i V_i^T$
- Set rotation $R_i = V_i U_i^T$
- Handle reflection:

$\boldsymbol{det(R_i) < 0}$	$\boldsymbol{det(R_i) \geq 0}$
$R_i = \begin{bmatrix} R_{i_{11}} & R_{i_{12}} & -R_{i_{13}} \\ R_{i_{21}} & R_{i_{22}} & -R_{i_{23}} \\ R_{i_{31}} & R_{i_{32}} & -R_{i_{33}} \end{bmatrix}$	$R_i = \begin{bmatrix} R_{i_{11}} & R_{i_{12}} & R_{i_{13}} \\ R_{i_{21}} & R_{i_{22}} & R_{i_{23}} \\ R_{i_{31}} & R_{i_{32}} & R_{i_{33}} \end{bmatrix}$

Optimizing Position (part 1)

For each vertex i, want to satisfy:

$$\sum_{j \in N(i)} w_{ij}\left(p_i' - p_j'\right) = \sum_{j \in N(i)} \frac{w_{ij}}{2}(R_i + R_j)(p_i - p_j)$$

Can be expressed as: $Lp' = b$

Each row i in L corresponds to one instance of the equation above:

- Diagonal $\mathrm{L_{ii}} = \sum_{j \in N(i)} w_{ij}$
- Off-diagonal $L_{ij} = -w_{ij}$
- If i, j are not neighbors, $L_{ij} = 0$

L is the famous Laplacian matrix.

Optimizing Position (part 2)

Solving $Lp' = b$ is done by factorizing $L = FF^T$ then solving.
This is possible because L is symmetric positive definite.

Constraints are implemented by setting rows to identity.
Problem: Setting constraints makes it no longer symmetric

$$\begin{bmatrix} L_{fxf} & L_{fxc} \\ 0_{cxc} & I_{cxc} \end{bmatrix} \begin{bmatrix} p'_f \\ p'_c \end{bmatrix} = \begin{bmatrix} b \\ V_c \end{bmatrix}$$

f: number of "free" vertices (unconstrained)
c: number of constrained vertices

Optimizing Position (part 3)

Treat matrix as blocks:

$$\begin{bmatrix} L_{fxf} & L_{fxc} \\ 0_{cxc} & I_{cxc} \end{bmatrix} \begin{bmatrix} p'_f \\ p'_c \end{bmatrix} = \begin{bmatrix} b \\ V_c \end{bmatrix}$$

Solve for $L_{fxf} p'_f$:

$$0_{cxc} p'_f + I_{cxc} p'_c = Vc$$
$$\Rightarrow \mathrm{p}'_\mathrm{c} = \mathrm{V}_\mathrm{c}$$
$$L_{fxf} p'_f + L_{fxc} p'_c = b$$
$$\Rightarrow L_{fxf} p'_f + L_{fxc} V_c = b$$

Leads us to the equation we *actually* want to solve:

$$L_{fxf} p'_f = b - L_{fxc} V_c$$

Since L_{fxf} is positive symmetric definite, solving it is efficient again.

Just had to fudge the right side of the equation before solving it.

Vertex weights

Well-known cotangent weights are used:

$$w_{ij} = \frac{1}{2}(\cot(\alpha_{ij}) + \cot(\beta_{ij}))$$

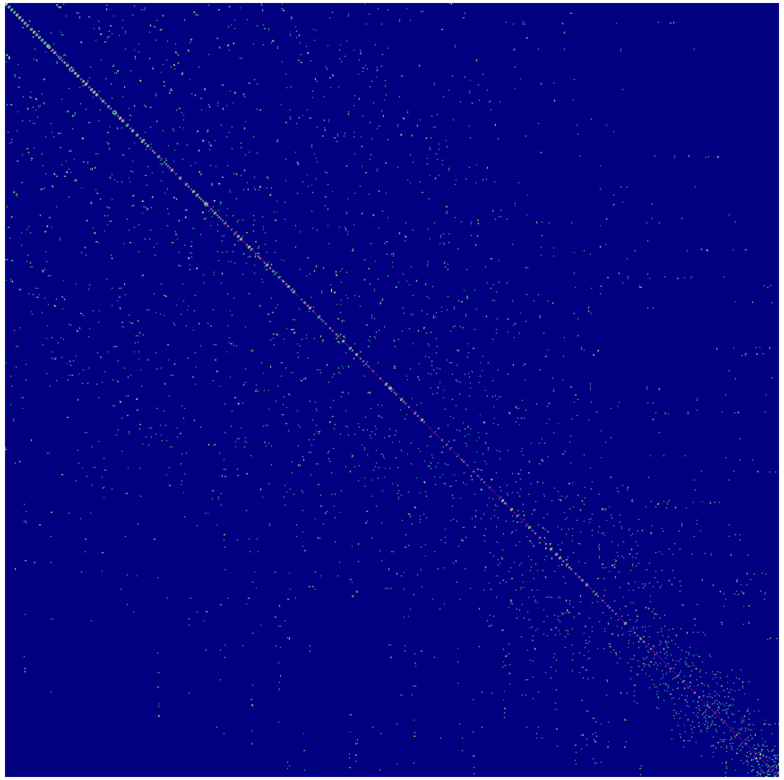
Note: cot is computed using $\frac{\cos}{\sin} = \frac{dot(u,v)}{||cross(u,v)||||u||||v||}$

Note: Cotangent weights can give negative values. Must handle this:

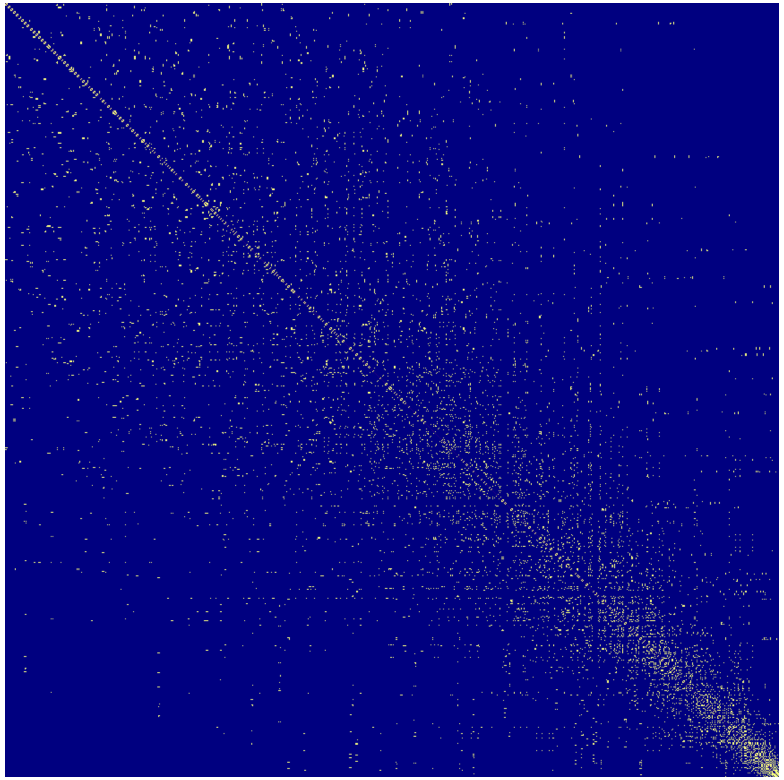
$\boldsymbol{w_{ij} < 0}$	$\boldsymbol{w_{ij} > 0}$
$w_{ij} = 0$	$w_{ij} = w_{ij}$

Sparsity patterns (armadillo)

Before factorization

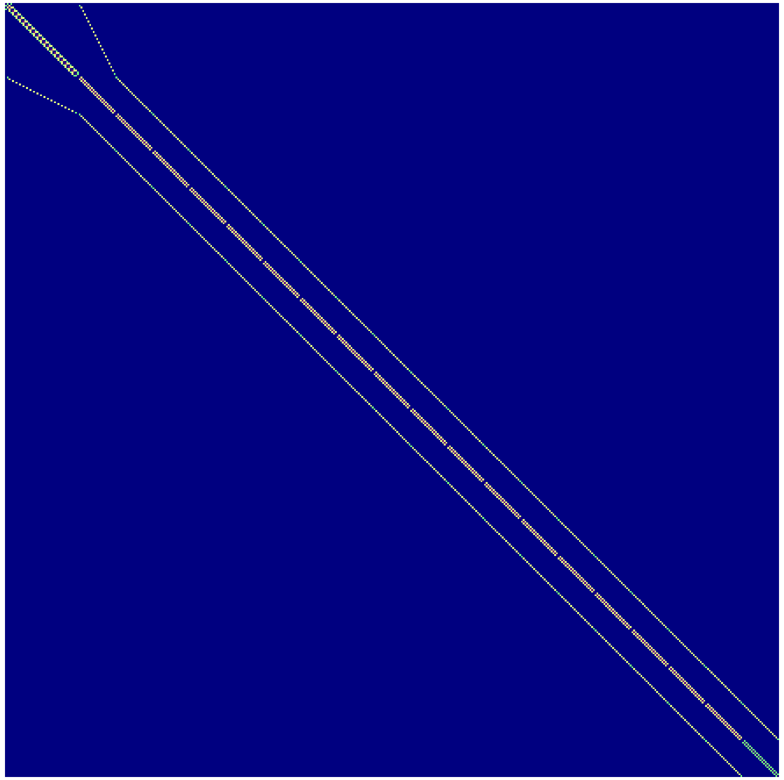


After factorization

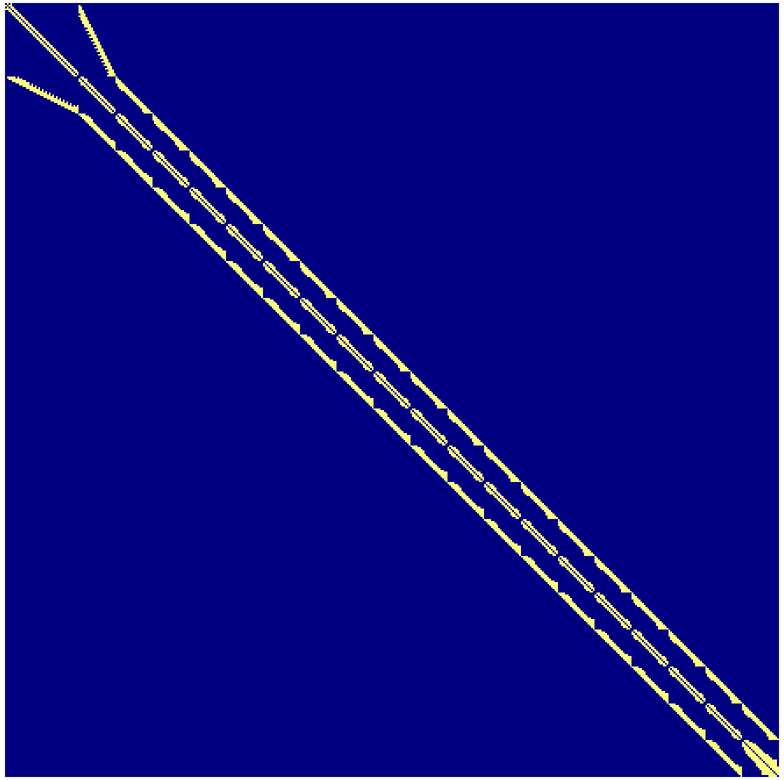


Sparsity patterns (square_21)

Before factorization

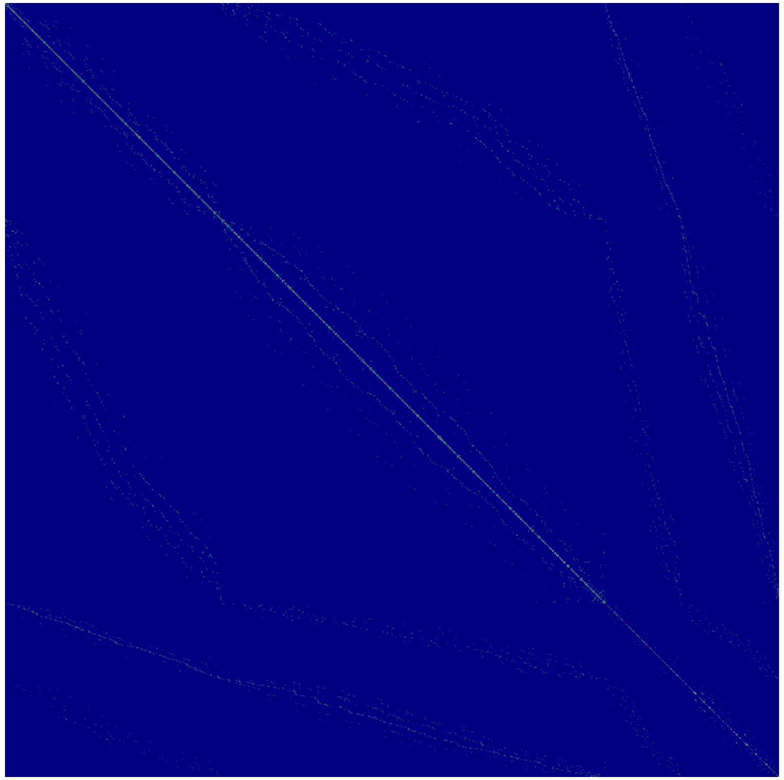


After factorization

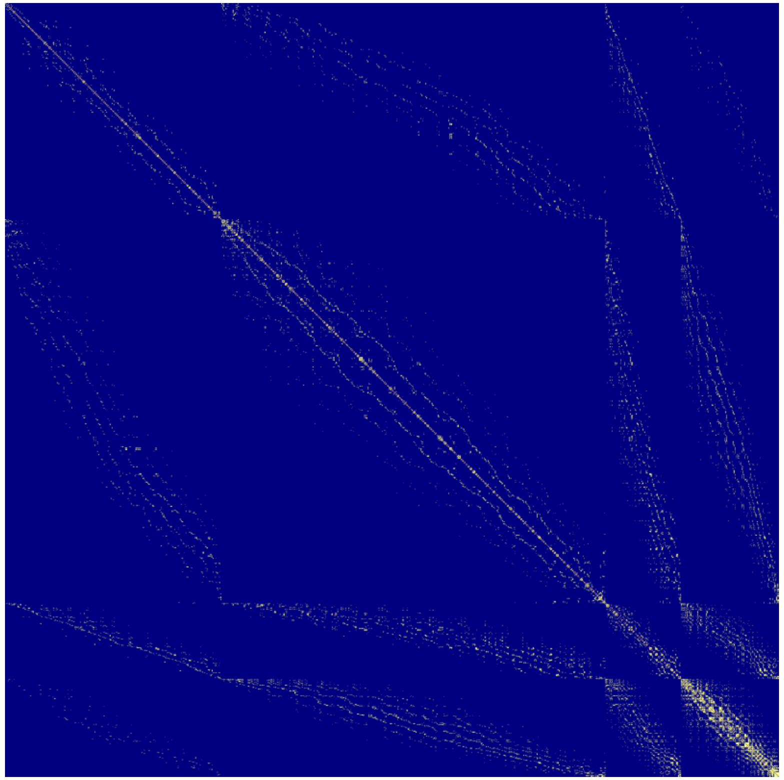


Sparsity patterns (cactus_highres)

Before factorization

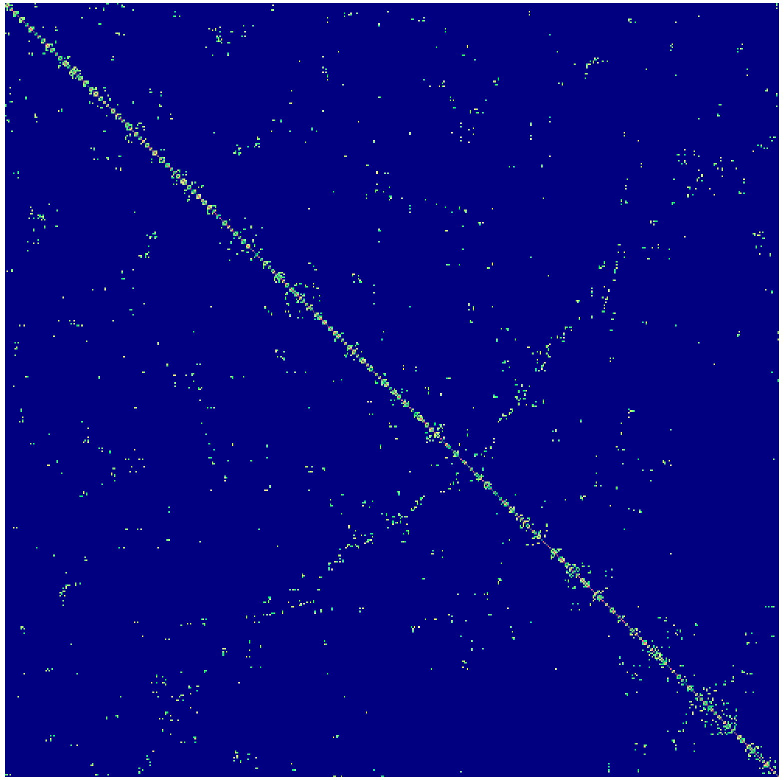


After factorization

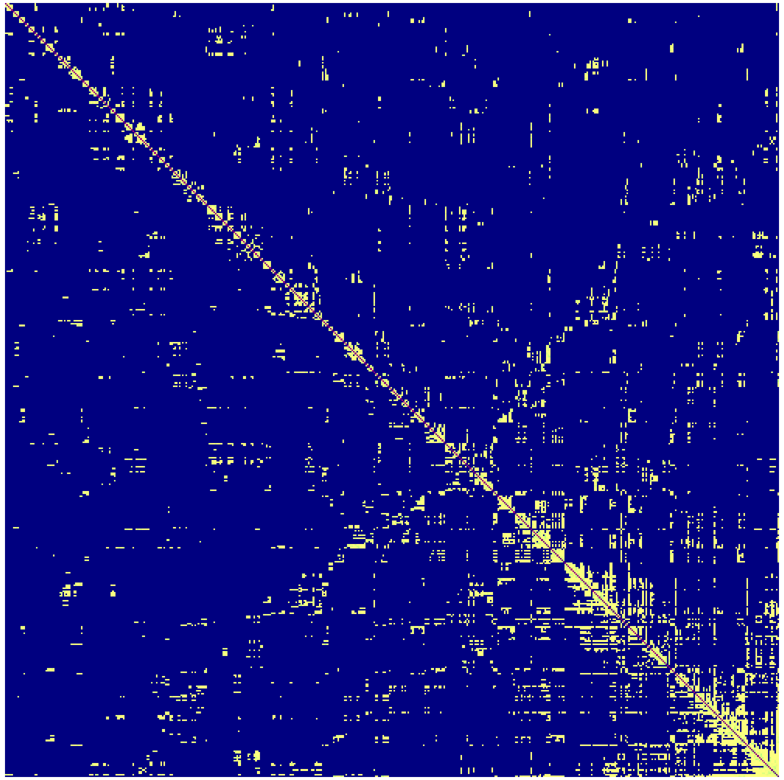


Sparsity patterns (indorelax's hand)

Before factorization



After factorization



Shortcomings

- Matrix solver is not sparse. Inefficient?
  - LAPACK doesn't support sparse matrices
  - Intel MKL's Sparse BLAS is obscure. Couldn't get it to do basic math. Broken?
- Didn't make a rotation widget (as shown in Sorkine, O.'s video)
- SVD3x3 solver used is very over-engineered (but fast apparently?)
- Storing and computing rotation matrices for ALL vertices.
  - Only actually need unconstrained vertices and their neighbors.
- (Rendering) SSAO is noisy because I haven't blurred it.
- (Rendering) No multisampling.

Upcomings

- Region-of-Interest polygon selection tool
- “Tutorial” GUI at bottom left gives step-by-step guidance
- (Rendering) “Ray traced” spheres for selection points
- (Rendering) Nice-looking line for the ROI selection
- (Rendering) SSAO
- “Interactive” with indorelax (~12k vertices)
  - Takes ~3 seconds to factorize matrix
    - (note: matrix sparsity images take a few more seconds to produce, if enabled)
  - Real-time if you select only part of indorelax with the ROI tool