CPSC 601.13 Final Project On Domand Discrete Element Synthesis in 3D

Timothy Davison
April 2014

***Abstract*—Discrete element textures methods can generate elements in a large output domain whose distribution is very similar to a smaller example. This is a task that is very tedious for a human user. Synthesizing discrete elements into an expanding 3D output domain, interactively just as they are needed, is an open question. This project builds on the approach of [2] to formulate a smaller optimization problem, with less iterations, and that can be used to expand the solution arbitrarily. However, future work is still needed in order to make it perform at interactive levels.**

I. Introduction

Many physical phenomena can be described by an underlying distribution. For example, consider skin, it consists of hair follicles, skin cells, pores and blood vessels. One patch of skin looks very similar to another patch of skin. Similar examples include swarms of fish, rocks on a beech, the bark of a tree, or even the buildings of a metropolis.

This project explores how to generate elements in a large domain just as they are needed based upon a small input example. Its inspired by seeding approaches such as [7], and the iterative optimization used in [2]. In the latter, elements in the output domain are matched to elements in the input domain. Those pairings then provide predictions for the attributes and positions of elements in the output domain, which can be solved in an optimization step. In their work, this is applied many times, and across the entire domain until a satisfactory result is obtained.

The key idea in this project is that optimization needs only to occur along, and in a small area around a so-called horizon. The horizon is a subset of the output domain containing newly created elements copied from the input example. Each round of the algorithm expands the domain and creates a new horizon. The optimization step is weighted such that the further away from the horizon an element is, the more locked in place it is, until at a certain distance its position and attributes are completely locked and no longer part of the optimization. In contrast to [2] each element only needs a few optimization steps.

In an attempt to ensure that few steps of optimization are needed, the algorithm attempts to choose good candidate elements for the horizon. This was done using Single Player Monte-Carlo Tree Search [1]. As will be discussed in the results section, this expensive combinatorial search is unnecessary, instead a patch-based approach will be discussed as the basis for future work.

A. Related Work

An image or texture is local if given a small window, neighbouring pixels or elements can be successfully predicted from other elements in the window [7], [4]. For example, in an image of sand, different windows into that image look very similar—the image has locality in the distribution of its pixels. This insight is the basis for texture synthesis.

Example based texture synthesis uses an input example to generate a larger texture that is non-repeating and that is locally similar to the example texture. Usually this is formulated as either a local or global optimization problem. Example based texture synthesis has been used to generate 2D textures [7], [8], [10], [11], [6], [12], 3D volumetric textures [5] and discrete distributions of elements in 2D and 3D space [3], [2].

For example, Kwatra uses a scan-line algorithm to identify which pixels to generate next, the colour of those pixels is selected from an input example whose neighbourhood is most similar to the neighbourhood of the output pixel [10]. An example of one neighbourhood similarity metric will be discussed shortly.

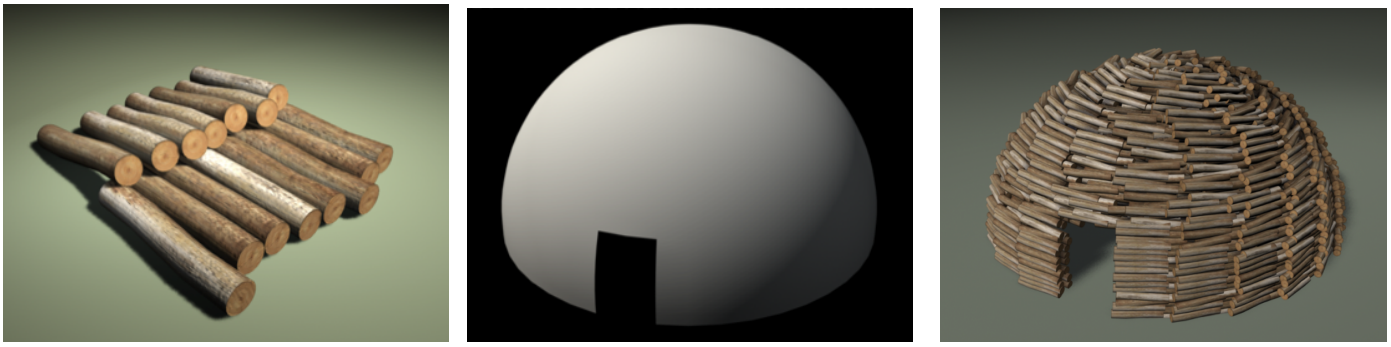
Patch-based synthesis is a natural extension of pixel-based synthesis where patches, instead of pixels, are copied from the input exemplar to the output domain [6], [8], [9]. The entire output-domain is then optimized such that it looks similar to the input domain.

B. Discrete Element Texture Synthesis

Discrete Element Textures (DET) is a technique that synthesizes non-repeating elements within a large domain based upon a small input example. The position and type of elements in the output domain are chosen using an optimization technique based upon the expectation-maximization algorithm. During the expectation step, the authors create a set of predictions for the type and position of elements in the output domain. In the maximization step, those predictions are used to optimize the position and type of elements so that their local neighbourhood is similar to the input example. These two steps are repeated until the systems converges on a solution. An example of this technique can be seen in Figure 1.

During the initialization phase sample patches from the input exemplar are randomly copied to the output domain such that all of the samples that belong to the same sample are copied. The size of the patches is proportional to a user defined neighbourhood extent. The next steps of the algorithm iteratively *search* for the most similar input neighbourhood for each output sample and then *assign* to that sample a position and attributes (such as element id and orientation).

For an output sample s_o and an input sample s_i the similarity between their neighbourhoods $|n(s_o) - (s_i)|^2$ is given by Equation 1. Intuitively this finds for output sample



(a) (b) (c)

Fig. 1: In this example of discrete element texture synthesis, the logs in a) together with the definition of the output domain in b) produce the set of synthesized elements in c).

s'_o in the neighbourhood $n(s_o)$ of the output sample s_o the most similar input sample s'_i in the neighbourhood $n(s_i)$ of s_i to s'_o. For each sample s' in the neighbourhood $n(s)$ of s we have the vector $\hat{p}(s') = p(s') - p(s)$. For each matching pair of samples $(s'_o \in n(s_o), s'_i \in n(s_i))$ in the neighbourhoods of s_o and s_i we find the difference between $\hat{p}(s'_i)$ and $\hat{p}(s'_o)$. This serves to align the neighbourhoods and compare the configuration of the positions of the samples in the input and output neighbourhoods.

$$|n(s_o) - n(s_i)|^2 = \sum_{s'_o \in n(s_o)} |\hat{p}(s'_o) - \hat{p}(s'_i)| + \alpha |q(s'_o) - q(s'_i)|^2 \tag{1}$$

During the search step *k-coherence* search is used to find the input sample s_i with the most similar neighbourhood $n(s_i)$ to the output sample's s_o neighbourhood $n(s_o)$. During the assignment step each sample $s'_o \in n(s_o)$ provides a prediction $\tilde{p}(s_o, s'_o) = p(s_i) - p(s'_i)$ for the relative position between s_o and s'_o. A least squares method is used to find the output positions that minimize the following energy function.

$$E_p(\{p(s_o)\}_{s_o \in \mathcal{O}}) = \sum_{s_o \in \mathcal{O}} \sum_{s'_o \in n(s_o)} |(p(s_o) - p(s'_o)) - \tilde{p}(s_o, s'_o)|^2 \tag{2}$$

To handle different domain shapes an additional energy term can be integrated into Equation 1. This term simply checks if a given sample is inside or outside of the given domain shape. The authors use a voxelization technique to perform this check.

Discrete element textures are not suitable for just-in-time synthesis for the following reasons: 1) The many-step optimization approach used in the paper is expensive. 2) Domains in DET are generated all at once. For just in-time-synthesis it would be desirable to expand the output domain as required. 3) New elements are added to the output domain only during the initialization step, this can lead to crowding and overlap (see Figure 2). There may exist a better approach to DET for just-in-time synthesis based upon different optimization techniques or by extending DET to address these points.

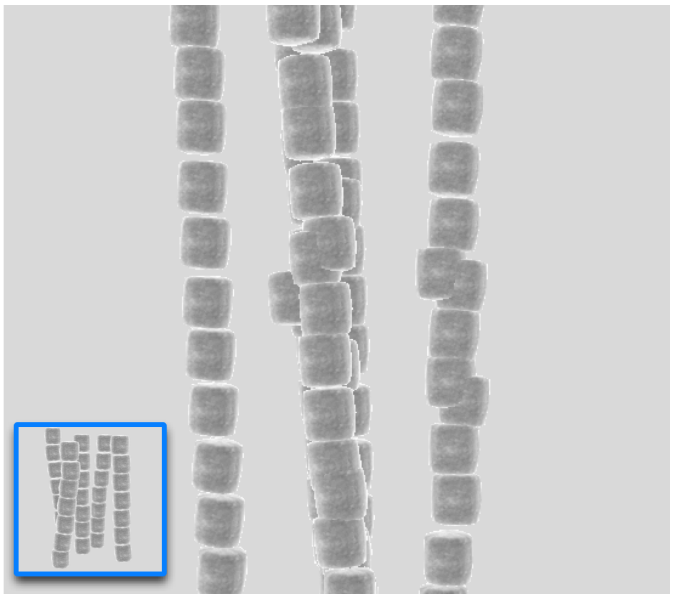


Fig. 2: An example of an output domain generated from an input domain (blue) where DET has produced overlap. The overlap occurs when the output domain is constrained and too many elements are added to it to satisfy the constraints imposed by the input example.

II. Representation

The position of an element s is denoted $p(s)$ while its attributes are denoted by $q(s)$. The vector $q(s)$ contains attributes such as the colour or the type of the element. In this paper we only allow boolean comparisons between elements of $q(s)$, for example $q(s_1)_t = q(s_2)_t$ might denote whether two elements share the same type. In this paper, the type determines the mesh of the element. It was not explored in this particular implementation, but we could also denote the rotation of an element with $r(s)$, right now rotation is just another attribute, but it could be optimized in the least-squares optimizer.

There are two domains, the output domain O and the example E domain. The horizon, $H \subset O$ contains the newly

created elements that need to be optimized.

This project measures texture similarity through a neighbourhood similarity metric. If $n(s)$ is the neighbourhood of s, then the distance between two neighbourhoods is given by $|n(s) - n(e)|^2$, which we define below:

$$|n(s)-n(e)|^2 = \sum_{s' \in n(s)} |(p((s)-p(s'))-(p(e)-p(e'))|^2 \quad (3)$$

Pair assignment in the above is unlike [2], the pairs $< s', e' >$ for $s' \in n(s)$ and $e' \in n(e)$ are determined in a greedy fashion. That is, the nearest pairs are assigned first, and removed from consideration. Ma et al. solve this using the Munkres assignment algorithm $O(n^3)$. We use a brute-force $O(\frac{1}{2}n^2)$ greedy algorithm. A k-d tree could reduce this to $O(log\ n)$. The Munkres assignment algorithm is optimal, however under observation, the pairings produced by the greedy algorithm produce good results.

A partial pair assignment is a set of pairs, $\{< s', e' > |s' \in n(s) \cup \emptyset, e' \in n(e) \cup \emptyset\}$. A partial assignment occurs if $|(p(s)-p(s'))-(p(e)-p(e'))|^2 < c$ for some user threshold. Partial assignments of the form $< 0, e' >$ indicate where to create new elements in the horizon (candidates).

Assignment, in general is the process of finding for an element $s \in O$ the element $e \in E$ whose neighbourhood $n(e)$ is closest to $n(s)$. This is done using a brute-force search. k-coherence search is not used because of partial assignment, instead future work would look at using a k-NN search in the example domain.

III. Algorithm

The proposed algorithm has two main phases. In the $generation$ step, new elements are added to the horizon through a combinatorial search of the candidate space using Single-Player Monte-Carlo Tree Search [1]. The candidate space is composed of the partial assignments for all elements in the previous horizon. The second part of the algorithm is an $optimization$ step. There are three components to this, in the first element positions are minimized in a weighted least squares approach, the next steps reassign and redistribute element attributes, $q(s)$.

In each round of the algorithm (generation and optimization), a new horizon is generated and the previous one discarded. This results in an expanding domain. Naturally synthesis can be constrained to a particular region by discarding candidates that fall outside of that region.

The goal of this algorithm is to first make good decisions that will start the horizon out in a low energy state. The energy is then reduced through least-squares optimization, reassignment, and redistribution steps. Furthermore, the size of the optimization problem is significantly smaller than in [2], as we need only consider elements in the horizon.

IV. Single-Player Monte-Carlo Tree Search

We use a standard Upper Confidence applied to Trees, Monte-Carlo Tree Search algorithm. UCT finds a balance between exploiting already explored and good areas of the tree, with the need to expand the search space with unexplored sections of the tree. Single-Player MCTS is an extension of MCTS from two-player win-loss games, to single-player scored games [1].

The number of ways to choose elements in the candidate set is huge $C(n, k)$, for n is the size of the candidate set, and k is the size of the horizon (which we don't know beforehand). Hence, MCTS seems like a good way to explore this space. It has the added advantage that it always tracks its best solution and can therefore be terminated at any point if the search is taking too long.

For any sequence of choices, we can measure the energy of the system by summing up the neighbourhood metrics for each element in the horizon, which gives us a score, i.e.:

$$score(H) = \sum_{h \in H} |n(h) - n(e)|^2 \quad (4)$$

where in this case $e \in E$ is the element in the example domain with the nearest neighbourhood to h.

Then, the decision of which node to choose at any level of the tree is to choose the child node with the highest:

$$\bar{x} + explore \cdot \sqrt{\frac{ln(n)}{n_i}} + \sqrt{\frac{\sum x^2 - n_i * \bar{x} + exploit}{n_i}} \quad (5)$$

Where $\bar{x}$ is the average playout score for the current node, n is the number of times the current node has been visited, n_i is the number of times the child node in question has been visited, and $\sum x^2$ is the sum of the results up until the current level of the tree (ie, a partial scoring up until the current decision). $explore$ and $exploit$ are user parameters to control how much MCTS should be exploring or exploiting the candidate space. This arrangement is maximizing, therefore we set x to be $2e_{max} - x$, where e_{max} was the maximal energy achieved in the last round of MCTS.

Each node stores its partial score, its average score, and the number of times its been visited. In addition it also stores its selected candidate, so that the candidate space at any particular level of the tree can be efficiently reconstructed without using too much extra memory.

As mentioned, candidates are produced by visiting the elements in the horizon and finding their nearest example, using the partial pair-assignment algorithm described earlier. Unassigned example elements are copied into the candidate space. For $h \in H$, the set of unassigned example elements are $< 0, e' >$ for $e' \in n(e)$ and e is the nearest example. Then the new candidate position $p_c = p(h) + (p(e) - p(e'))$. The attributes of $q(e')$ are simply copied.

To reduce the size of the candidate space, two filter operations are used. The most obvious is to eliminate overlapping candidates within a certain small radius.

The other candidate filter aims to make choices that will not increase the energy of the system significantly. For a given candidate c, we find the nearest example element e. If $|n(c) -$

$n(e)|^2 > t|$, where t is 1.5 times the average neighbouring assignments.

A round of MCTS terminates when there are no candidates left. The score is then back-propagated.

V. Optimization

Optimization consists of three parts: a least-squares optimization of element positions, attribute reassignment and attribute redistribution.

A. Least-Squares Element Position Optimization

In the previous step, the horizon has been filled with elements. For each $h \in H$ we find the nearest $e \in E$ using the full-pair-assignment algorithm. Then for each $h' \in n(h)$ we have a set of pairs $\{< h', e' >\}$ for $e' \in n(e)$ that predict positions for h, that is

$$\hat{p}(h) = (p(s) - p(s')) - (p(e) - p(e')) \tag{6}$$

As in [2] we want to minimize the energy of the elements in the horizon in the manner of Equation 6. Consider the following example:

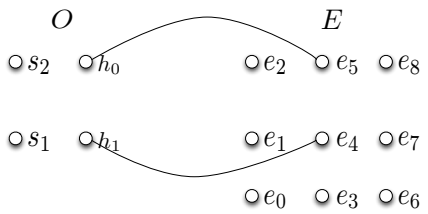


Fig. 3: The left shows elements in the output domain, while the right shows elements in the example domain. The curves identify the example elements assigned to the horizon elements.

We can form a system of equations for the example where the pairings in the neighbourhoods of h_0, h_1, e_5, e_4 provide predictions for the positions of h_0, h_1, s_1, s_2:

$$\begin{aligned}
h_0 - h_1 &= e_5 - e_4 \\
h_0 - s_1 &= e_5 - e_1 \\
h_0 - s_2 &= e_5 - e_2 \\
h_1 - h_0 &= e_4 - e_5 \\
h_1 - s_1 &= e_4 - e_1 \\
h_1 - s_2 &= e_4 - e_2 \\
h_0 &= h_0 \\
h_1 &= h_1 \\
w_1 \times s_1 &= w_1 \times s_1 \\
w_1 \times s_2 &= w_1 \times s_2
\end{aligned} \tag{7}$$

The right side of this system of equations forms a vector b, for each component x, y, z. The left side forms a matrix A and unknowns $x = (w_0 \times h_0, w_1 \times h_1, s_1, s_2)^T$. Here we see the weighted component of the formulation: the elements outside of the horizon, but still within the neighbourhood distance of its elements. The weight given is proportional to the element's distance from the horizon. Hence the system of equations will try to optimize the horizon over the already optimized values.

The system of equations can be reduced to just 4 rows, hence we end up with a 4×4 A matrix, or in general an $n \times n$ matrix for a horizon of size n. This system of equations, $Ax = b$ is solved using the Cholesky decomposition module of the Eigen matrix library. This is a very high performance and widely used implementation that uses x86_64 SSE2 instructions for SIMD vector operations.

A constrained least squares formulation was also tried. However, with QR decomposition it was significantly slower, and produced inferior results. Thinking about this, the weighted solution allows the newly introduced elements and existing neighbours to reach a better configuration with respect to one another, and hence the system can reach a lower energy configuration.

B. Reassignment

The nearest example for each horizon element from the least squares step is stored, along with all of the pairs with examples unassigned with output elements, $\{< 0, e' >\}$. If there is an s' such that $|(p(s) - p(s')) - (p(e) - p(e'))|$, regardless of whether $q(e') = q(s')$, then the attribute vector for $q(e')$ provides a vote to change each attribute in $q(s')$. Attributes of the element s' are changed if enough votes are cast (greater than some user constant). The reassignment step allows elements to be assigned new attributes after the MCTS generation step, this is necessary as the surrounding neighbourhood may have changed significantly since the initial assignment.

C. Redistribution

One goal of achieving texturing similarity is that the frequency of elements in two local neighbourhoods should be the same. To this end, for element $h \in H$ the local distribution of attributes $q(h'), h' \in n(h)$ forms a histogram. Then, for the matching example element $e \in E$ to h its histogram is also found. If $q(h)$ is overrepresented relative to its corresponding example's histogram, then it is reassigned to the most underrepresented frequency in h's histogram. In addition to reassignment, re-distribution of attributes helps achieve texture similarity in terms of element attributes.

VI. Discussion and Results

For example domains with regular structure, the approach in this paper is very fast and stable. In particular, the result in Figure 4 took about 5 seconds of algorithm time running on 3 threads and 4 seconds of time in the simulation environment to load all of the graphical resources and update the user interface. In particular, the small example domain size hides the complexity of the assignment step. In later results we'll see how expensive this step can be.

In Figure 5 the same example domain from Figure 4 is used. This time, however, least squares optimization is not applied. One can see the stability that results from a single MCTS play-out. This suggests that full MCTS isn't need and that a simple filtering strategy as described in Section IV is

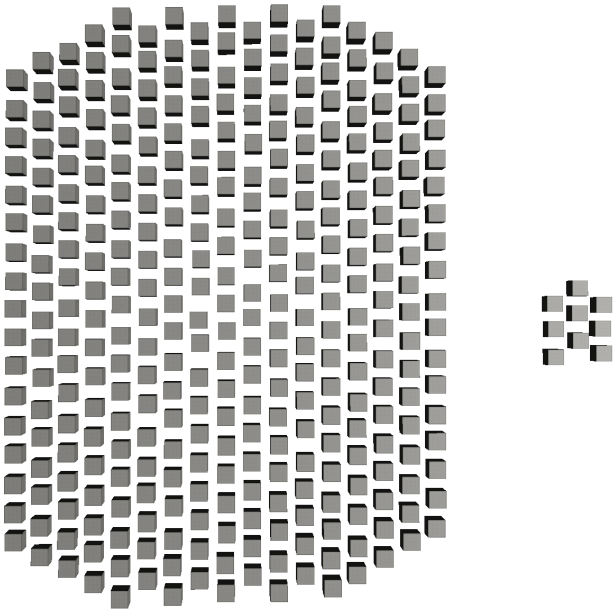


Fig. 4: The elements to the right are the example domain while the elements to the left are the output domain. The example domain is slightly staggered to illustrate the stability of the approach. Algorithm time: 5s.

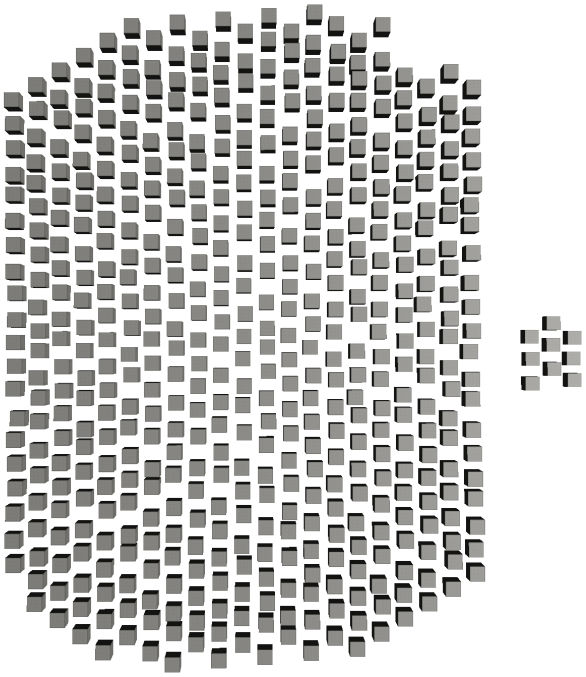


Fig. 5: Repeated application of only the generation step without optimization. Algorithm time: 3s.

sufficient. Running 20 rounds of MCTS on Figure 4 results in a visually identical result, and took 89 seconds, see Figure 6.

In the next set of examples, we explore the synthesis of a patch of cellular membrane. The blue spheres with orange legs are phospholipids, while the purple tubes are protein channels, the orange spheres are small molecules that can travel through the protein channels. Each example only required one round of MCTS. Each lipid is approximately 1.0 units away from its neighbours. For both results below, the example in Figure 7 was used.

First, a small neighbourhood was used as in Figure 8. As one can see this produced a result that did not include the orange molecules. Furthermore, the lipids overlap and penetrate with the protein channels.

In the next example, the neighbourhood distance was increased to 4.0 units. This significantly improved the results as one can see in Figure 9. In particular there is less inter-penetration, and the orange molecules are correctly synthesized. Furthermore, the results of the generation steps were not significantly different after applying the optimization steps. My interpretation is that the examples chosen for each output element are local to each-other within the example domain. Thus, they have more coherence in the output domain and hence less energy in the output domain. This again suggests that MCTS may not be needed as one can simply increase the neighbourhood size. Increasing the neighbourhood size increases complexity, a patch based approach could probably achieve the same result as a large neighbourhood and MCTS play-out, but with significantly less cost.

Another point to notice is that the orange molecules are

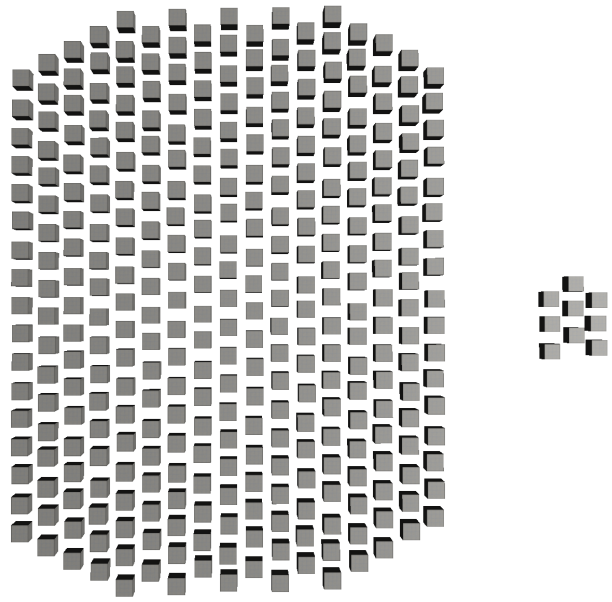


Fig. 6: Repeated application of generation and optimization. Each generation step involved 20 rounds of MCTS. Algorithm time: 89s.

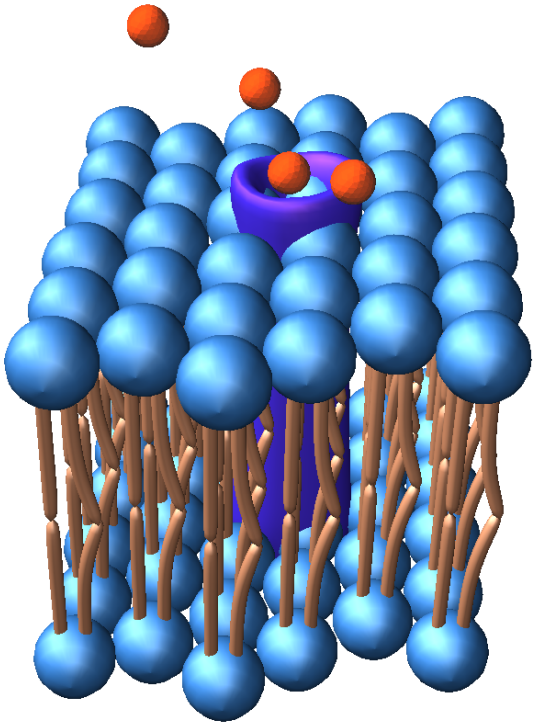


Fig. 7: A cellular membrane example. The blue spheres with orange tails are phospholipids, the orange spheres small molecules, and the purple tubes are protein channels.

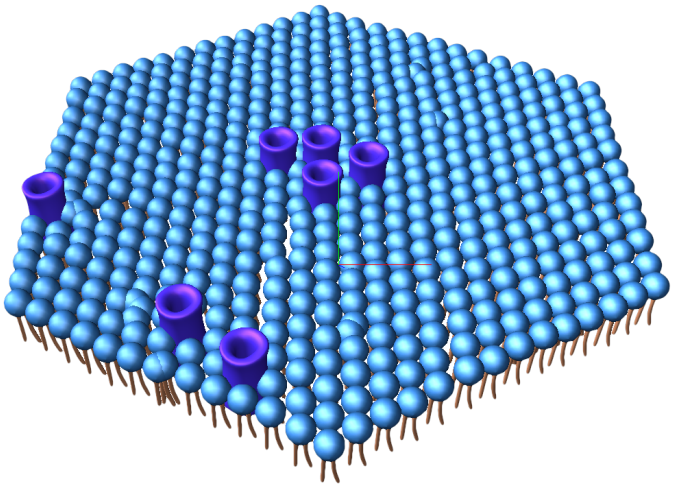


Fig. 8: The result of using a neighbourhood of radius 1.5 The orange molecules are missing and there is a lot of penetration and overlap.

overrepresented, even with the redistribution and reassignment steps. This is probably an artifact of the elements being widely spaced apart.

Notice the floating lipid. The orange molecules "sucked" it out of the membrane in the least-squares optimization. Also notice that there is still penetration of the protein channels as well as empty regions where there should be protein channels. To compensate for these artifacts, perhaps the distance metric should be weighted by distance but also by frequency of element attributes. The protein channels occur with low frequency, hence the effect of the far more populace lipids is to crowd them. A distance metric that accounts for the frequency of the protein channels could give their direction vectors in the least squares optimization more weight.

The relatively large example combined with the large neighbourhood distance (which works out to about 25 elements in a neighbourhood) dramatically increased the running time for Figure 9, namely more than 20 minutes. For Figure 8 that was about 4 minutes.

VII. Conclusion and Future Work

Most of the performance issues in the results section are the result of using poor algorithms when better ones are available. In particular, k-d trees could be used to accelerate pair-assignment and nearest example assignment. For instance, the nearest example assignment is $O(l*m*n^2)$. Where, n is the number of neighbours, m the example size, and l the horizon size. Without considering the attributes of an element and only considering element position, one can use a k-d(imensional) tree, where k is three times the neighbourhood size. The search complexity is on average $O(logn)$ or $O(n)$ in the worst case. However, one also has to consider the attributes of the elements in determining the nearest example to an output element. This will be one area of study over the coming weeks. I will start by looking at k-nearest-neighbour search.

Improving the partial-pair assignment could incorporate the Earth Mover's Distance metric [13]. In addition, the neighbourhood distance metric could account for attribute frequency in the local neighbourhood. At the very least, attribute frequency should be added as a weighted term to the least-squares solver.

As was discussed in the results section, larger neighbourhoods gave better results, especially with few steps of MCTS. Therefore, in the next phase of research, I will abandon MCTS and explore using patch based synthesis. In this scheme, one would choose patches such that there are no uncovered regions. For covered domains especially, this would require overlap between patches. The overlapping and boundary regions would form the horizon. Then, we can apply the filtering steps from Section IV to the horizon, followed by the optimization steps. Choosing patches could be done randomly, if that turns out to be a poor choice, then patches could be chosen such that their boundaries or overlaps lead to the least increase in energy in those regions.

In conclusion, the algorithm outlined in this paper is very performant for small examples, however, the implementation is slow for larger example domains. This can be fixed. The MCTS step is a poor fit for the goals of this project, namely to find an approach that can produce high quality output domains

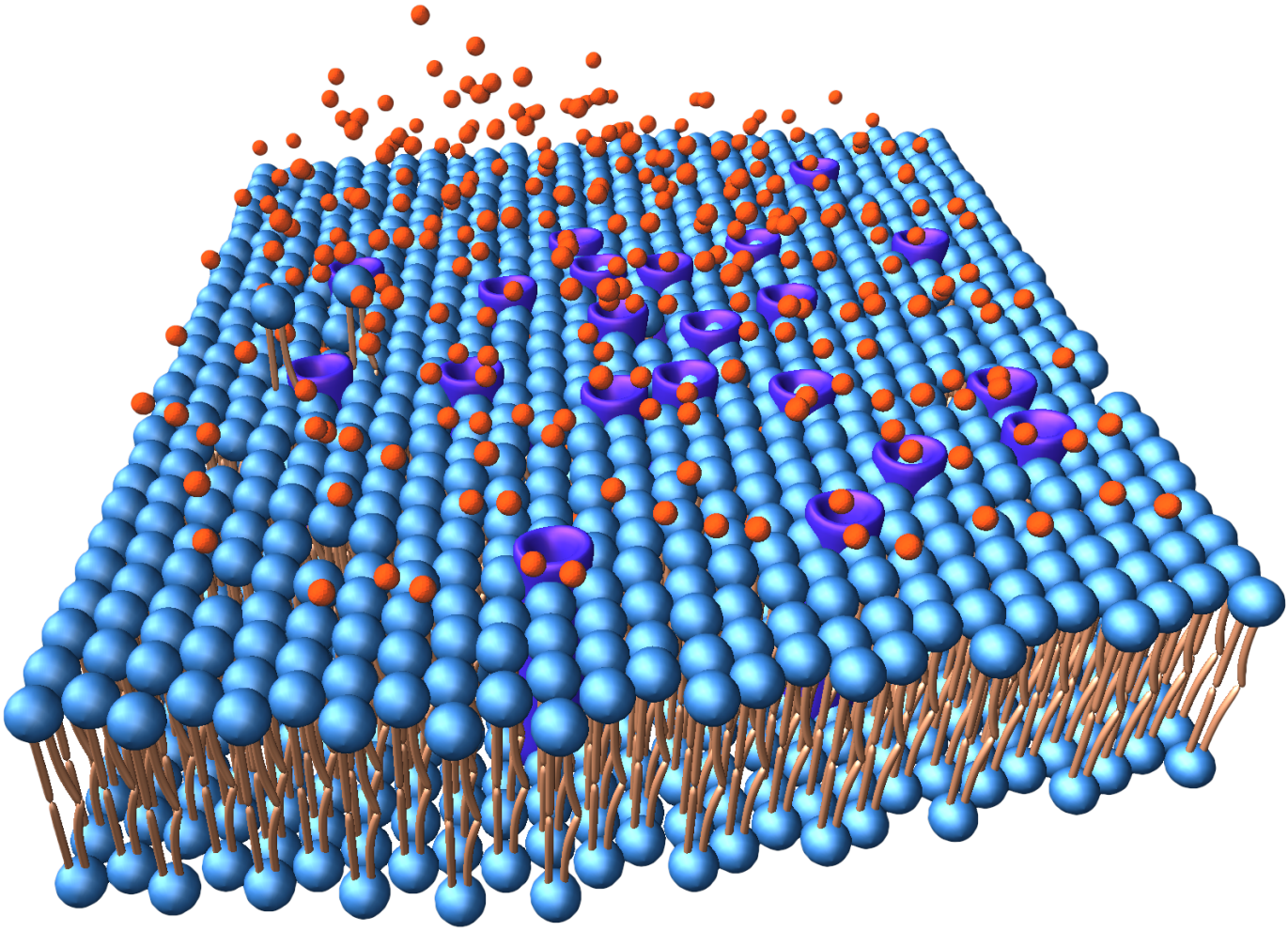


Fig. 9: A neighbourhood radius of 4.0.

in a minimum amount of time. Ideally, the system should be something that can run in real time. The two approaches outlined above will build on the work and results already achieved and I believe will result in an approach that meets the goals of this project (and a nice publication).

REFERENCES

[1] M. P. D. Shcadd, M. H. M. Winands, M J. W. Tak, J W. H. M. Uiterwijk, "Single-player Monte-Carlo tree search for SameGame", *In Knowledge-Based-Systems*, 32, (2012), pp. 3–11.

[2] C. Ma, L. Wei, X. Tong, "Discrete Element Textures", *In SIGGRAPH '11*, (2011), article 62.

[3] L. Wang, Y. Shi, Y. Chen, and P. Voicu, "Just-in-Time Texture Synthesis", *In Computer Graphcis Forum*, 32, 1 (2013), pp. 126–138.

[4] K. Vanhoey, B. Sauvage, F. Larue, J. M. Dischler, "On-the-fly multi-scale infinite texturing from example" *AACM Transactions on Graphics*, 32, 6 (2013), Article 208.

[5] S. Du, S. Hu, R. Martin, "Semiregular Solid Texturing from 2D Image Exemplars" *IEEE Transactions on Visualization and Computer Graphics*, 19, 3 (2013), pp. 460–469.

[6] E. Praun, A. Finkelstein, "Lapped Textures", *In SIGGRAPH 00*, (2000), pp. 465–470.

[7] L. Wei, M. Levoy, "Fast texture synthesis using tree-structured vector quantization", *In SIGGRAPH '00*, (2000), pp. 479–488.

[8] L. Liang, C. Liu, Y. Xu, B. Guo, H. Shum, "Real-time texture synthesis using patch-based sampling", *In ACM Transactions on Graphics*, 20, 3 (July 2001), pp. 127–150.

[9] Z. AlMeraj, C. Kaplan, P. Asente, "Patch-based geometric texture syntehsis", *CAE '13, In Proceedings of the Symposium on Computation Aesthetics*, (2013), pp. 15–19.

[10] V. Kwatra, I. Essa, A. Robick, N. Kwatra, "Texture optimization for example-based synthesis", *In SIGGRAPH '05*, (2005), pp. 795–802.

[11] X. Tong, J. Zhang, L. Liu, X. Wang, B. Guo, H.-Y. Shum, "Synthesis of bidirectional texture functions on arbitrary surfaces", *In SIGGRAPH '02*, (2002), pp. 665–672.

[12] L. Wei, S. Lefebvre, V. Kwatra, G. Turnk, "State of the art in example-based texture synthesis", *In Eurographics 09 State of the Art Report*, (2009), pp. 93–117.

[13] Y. Rubner, C. Tomasi, L. J. Guibas, "The Earth Mover's Distance as a Metric for Image Retrieval", *International Journal of Computer Vision*, 40, 2 (November 2000), pp. 99–121.