Euclidean Distance Transform

The distance transform of a binary (2D or 3D) picture P labels each object pixel or voxel (value is 1) with the distance of this pixel or voxel to the nearest non-object pixel or voxel (value is 0; background is assumed to have value 0). For all pixels or voxels p of P, the algorithm determines

$$t(p) = min_k\{d(p, q_k) : P(q_k) = 0 \wedge 1 \leq k \leq S\}$$

where $d(p, q_k)$ denotes a metric, and k lists all S pixels or voxels in the picture. It follows that $t(p) = 0$ for all non-object elements. Obviously, the values for $t(p)$ depend on the chosen metric.

For the *squared Euclidean distance transform* (SEDT) we substitute $d(p, q_k)$ by $d_e^2(p, q_k)$.

For the *Euclidean distance transform* (EDT) we substitute $d(p, q_k)$ by $d_e(p, q_k)$.

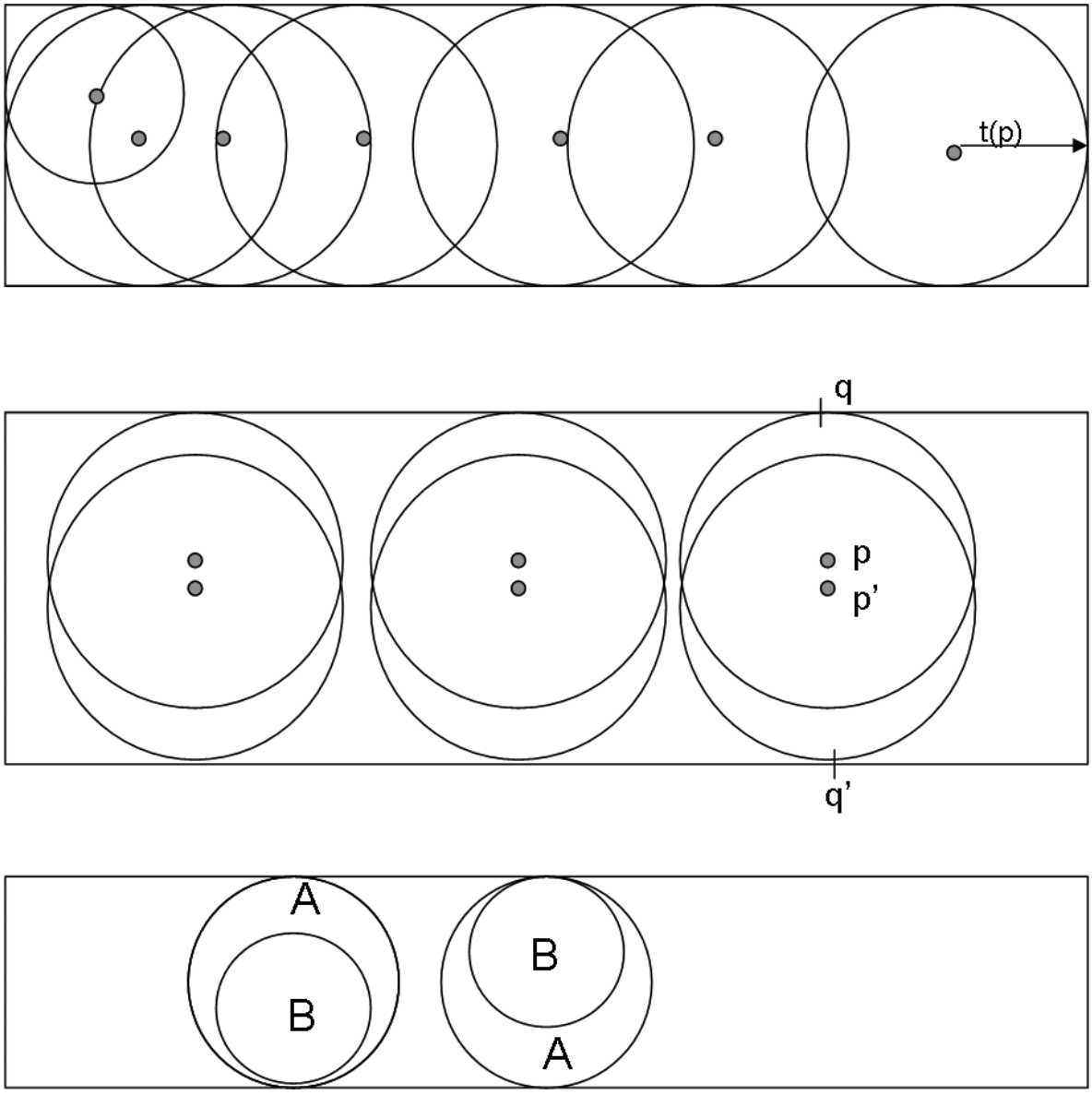
We recall that the Euclidean metric is as follows:

$$d_e(p, q) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2}$$

for points $p = (x_1, x_2, ..., x_n)$ and $q = (y_1, y_2, ..., y_n)$. The SEDT allows to perform all calculations in integer format.

We start with $n = 2$. We will then show that the SEDT or EDT easily generalizes to arbitrary $n \geq 2$, allowing to consider 4D, 5D, and so forth binary "pictures" as well.

The notes for this lecture are based on publications by [Saito and Toriwaki 1994], [Hirata 1996], [Meijster et al. 2000], [Toriwaki and Mori 2001], and [Bailey 2004]



Top: a rectangle with a subset of maximal disks. Middle: maximal disks in a rectangle where two centers of maximal disks are adjacent to each other. Bottom: disk B has only one non-object pixel at distance r to the center of B, and B is totally inside the maximal disk A.

Properties

Independent from the chosen metric, for given (not necessarily connected) sets M of object elements, and B of non-object elements, the distance transform has the following properties:

1. $t(p)$ represents the radius of the largest disk or ball centered at p and totally contained in M (top row in figure).
2. If there is only one non-object element $q \in B$ with $t(p) = d(p, q)$ then there are two cases:
   (a) an element $p' \in M$ exists such that the disk or ball centered at p' totally contains the disk or ball centered at p (bottom row in figure; disk B is totally included in disk A), or
   (b) elements $p' \in M$ and $q' \in B$ exist such that $d(p, q) = d(p', q')$ and p is adjacent to p' (middle row in figure).
3. If there are two (or more) non-object elements $q, q' \in B$ such that $t(p) = d(p, q) = d(p, q')$, then the disk or ball centered at p is a maximal disk in M and p is symmetric (see top row in figure)

Distance Transform Algorithms

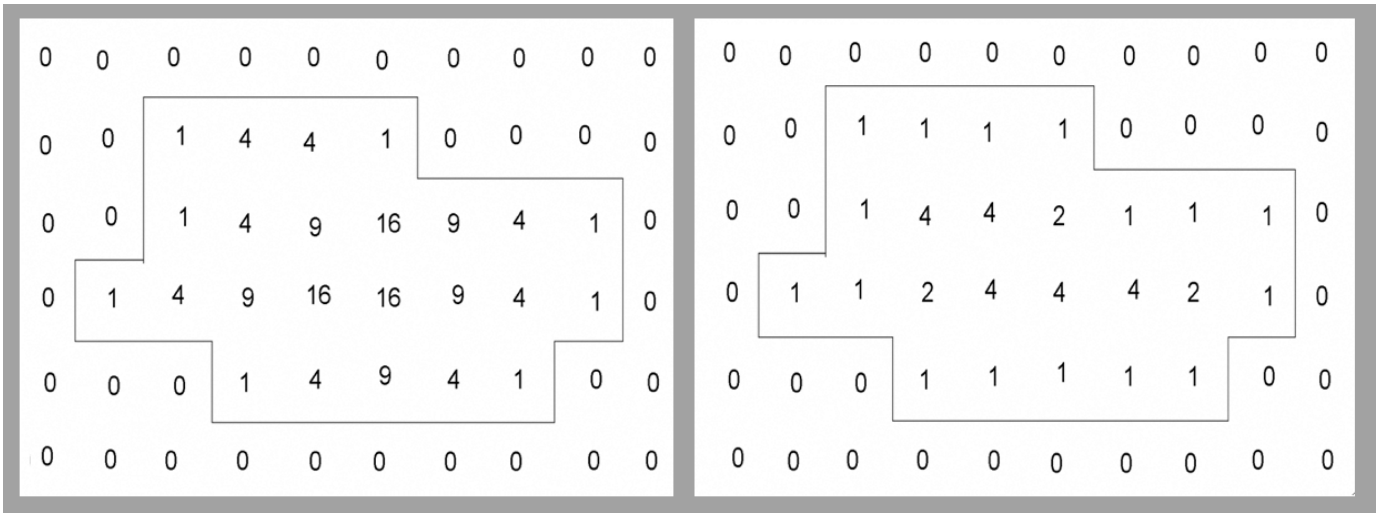
The *distance map* is a 2D array (3D for 3D pictures) of the same size as the original picture which stores the results $t(p)$ for all elements p. Algorithms for efficient approximation or computation of Euclidean distance maps can be classified as follows:

1. **Two path algorithms** compute approximations of the EDT in two passes (forward raster sequence and reverse raster sequence, linear time).
2. **Vector propagation algorithms** compute EDT in linear time, but values are incorrect in a small percentage of cases.
3. **Iterative distance transform algorithms** compute approximations of the EDT in iterations. Each iteration computes only the distance from border elements to non-object elements depending on selected pixel adjacency (linear time).
4. **Envelope algorithms** compute exact Euclidean distances for arbitrary dimensions (linear time is possible).

In this lecture we discuss 2D envelope algorithms, and also generalizations for arbitrary dimensions. – Initial step: Calculation of distances, from object pixel positions $p = (x, y)$ to the nearest non-object pixel position in the same row:

$$f_1(x, y) = f_1(x - 1, y) + 1 \text{ if } P(x, y) = 1$$
$$f_1(x, y) = 0 \text{ if } P(x, y) = 0$$
$$f_2(x, y) = \min\{f_1(x, y), f_2(x + 1, y) + 1\} \text{ if } f_1(x, y) \neq 0$$
$$f_2(x, y) = 0 \text{ if } f_1(x, y) = 0$$

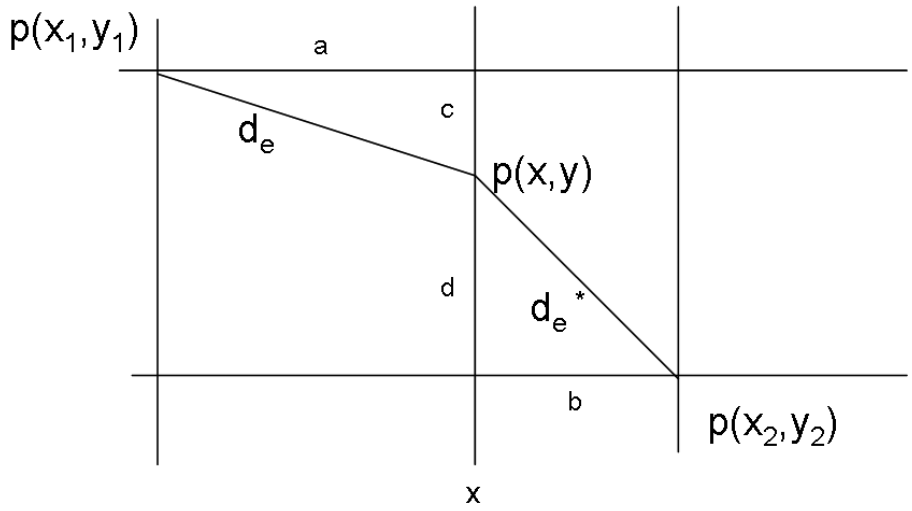
f_1 determines the distance between pixel position $p = (x, y)$ and the left nearest non-object pixel position q; f_2 replaces f_1 if the distance to the right border is shorter. The result is a matrix which stores integer values $(f_2(x, y))^2$ at each pixel position:



Left: result after row scans. Right: results after column scans.

This initial step has run-time in the order of MN, for M columns and N rows.

Computation of $t(x, y)$ for a fixed column x:



We assume that we have only two non-object pixels at $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, and we like to compute $t(x, y)$ for all y in a fixed row x (see above). We know that $a = f_2(x, y_1)$ and $b = f_2(x, y_2)$. We only need to decide whether element (x, y) is closer to p_1 or closer to p_2. That means we compare both values $(d_e)^2$ and $(d_e^*)^2$ to find the minimum, or (in other words)

$$t(x, y) = \min\{[f_2(x, y_1)]^2 + (y - y_1)^2, [f_2(x, y_2)]^2 + (y - y_2)^2\}$$

In general, for N rows we compute:

$$t(x, y) = \min\{[f_2(x, y_j)]^2 + (y - y_j)^2 : 1 \leq j \leq N\} \qquad (1)$$

Because x is fixed, we define $g(y_j)^2 = [f_2(x, y_j)]^2$ and rewrite Equation (1) as follows:

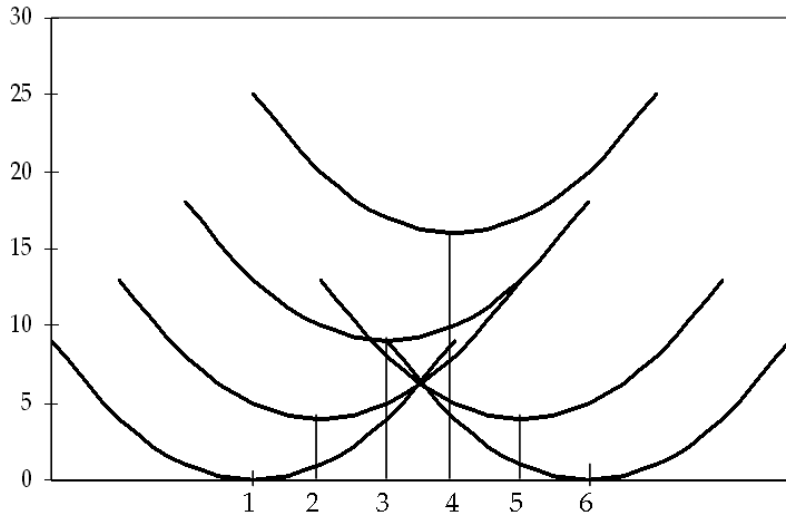
$$t(y) = \min\{g(y_j)^2 + (y - y_j)^2 : 1 \leq j \leq N\} \qquad (2)$$

For each y (that means N times) we compute N values, and obtain $t(x, y)$ (i.e., the number of calculations per column is in the order N^2).

For a more efficient implementation we start with a geometric interpretation. For simpler notation, y_j corresponds to row number j, and we replace y_j by j. For a fixed column (i.e., x is constant and $g(j) = f_2(x, j)$) and a fixed row (i.e., y is constant), the equation

$$\gamma_y(j) = [g(y)]^2 + (y - j)^2, \text{ for } j = 1, \ldots, N,$$

is a discrete parabolic arc. For $1 \leq y \leq N$, we consider a family of N parabolic arcs (see below), one arc for each row. Note that the horizontal axis represents the row number y, and the vertical axis represents $\gamma_y(j)$, with the local minima at $y = j$, and $\gamma_y(j) = g(j)$ in this case.



Parabolic arcs for column [0,4,9,16,4,0] in figure (left) on Page 5.

The *lower envelope* of the family of parabolic arcs corresponds to the minimum calculation in Equation (2), and vice versa.

The figure on Page 7 shows a family of six parabolic arcs. The lower envelope consists of two curve segments. The first segment starts at $(1, 0)$ and ends at the intersection of the first and the last parabola. The second segment begins at the intersection and ends at $(6, 0)$. Only two of six parabolic arcs contribute to the lower envelope of the family. To calculate $t(x, y)$ (x fixed) we determine the begin and the end of each contributing section and the index of the associated parabolic arc in one scan.

If y is between the begin and the end of a section then we use the associated parabolic arc to compute t. For example, when we compute $t(5, 3)$ then we need to know that $1 < 3 < 3.5$ belongs to the first section and the associated parabolic arc in the lower envelope is γ_1. Then

$$\gamma_1(3) = (g(1))^2 + (1 - 3)^2 = 4$$

Let y_s be the abscissa of the intersection and let $y_1 < y_2$. From the equation

$$[g(y_1)]^2 + (y_s - y_1)^2 = [g(y_2)]^2 + (y_s - y_2)^2$$

for the intersection $y_s = y_s(\gamma_1, \gamma_2)$ of two parabola γ_1 and γ_2, we derive

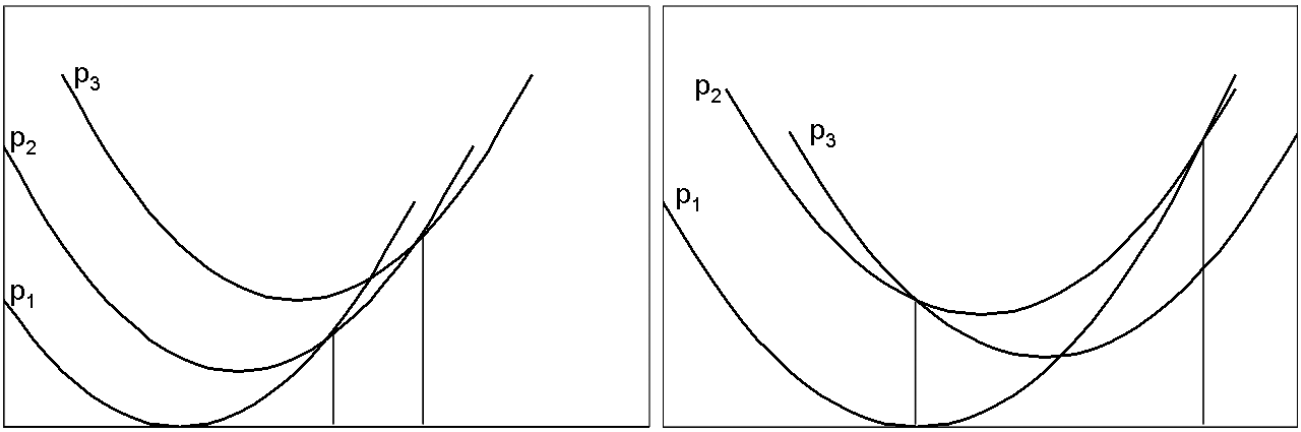
$$y_s = y_2 + \frac{[g(y_2)]^2 - [g(y_1)]^2 - (y_2 - y_1)^2}{2(y_2 - y_1)}$$

or

$$y_s = \frac{[g(y_2)]^2 - [g(y_1)]^2 + y_2^2 - y_1^2}{2(y_2 - y_1)}$$

Informal Description of Algorithm

We use a stack: only parabolas which contribute to the lower envelope stay in the stack; all the others are eliminated from the stack. A straightforward algorithm is the following:



Left: $y_s(\gamma_2, \gamma_3) > y_s(\gamma_1, \gamma_2)$. Right: $y_s(\gamma_2, \gamma_3) < y_s(\gamma_1, \gamma_2)$.

Each stack item stores a pair of real values (b, e) for the begin and the end of the section of a parabola which contributes to the lower envelope. (b_t, e_t) belongs to the top parabola of the stack and (b_f, e_f) is the pair associated with the following parabola in the sequential process. The first item stores begin and end of the section for the first parabola. It is initialized with $(1, N)$; the lower envelope would consist of one segment only if all the following parabolas do not generate intersections between $(1, N)$. Note that the parabolas are ordered according to their y-values in the picture. For each sequential step we evaluate the intersection for the top item of the stack representing γ_t and the next following parabola γ_f.

There are only three possible cases:

1. $y_s(\gamma_t, \gamma_f) > n$: γ_f does not contribute to the lower envelope; do not change the stack; go to the next parabola.
2. $y_s(\gamma_t, \gamma_f) \leq b_t$: Pop γ_t from the stack; evaluate the intersection of the new top item with γ_f (see right of figure on Page 9); if the stack is empty, add the item for γ_f to the stack.
3. $y_s(\gamma_t, \gamma_f) > b_t$: Adjust γ_t with $e_t = y_s(\gamma_t, \gamma_f)$; add the item for γ_f to the stack, with $b_f = e_t, e_f = n$ (see left of figure).

The procedure continues until the final parabola has been compared with the top item of the stack. Finally, only sections of the lower envelope are registered in the stack, and they are used for calculating the values for $t(x, y)$ in an additional scan.

Time Complexity

The initialization requires time in the order MN.

Then we have two runs through each column: one for calculating parabolic arcs and updating the stack, and one for calculating the final minimum distances. This makes M times order of N, that means again order of MN.

Because the problem size is $n = MN$, we have altogether a linear time algorithm with time complexity $\mathcal{O}(n)$.

Pseudo-code of Algorithm

The first step (initialization) has been finished already.

In a *forward scan* (top to bottom) we compute the sections of parabolas which contribute to the lower envelope. In a *backward scan* (bottom to top) we compute the distance values.

```
begin
    for x := 1 to M do begin
        q := 1; s(1) := 1; t(1) := 1;        (initialize values)
        for j := 2 to N do begin   (forward scan)
            while q > 0 and γ_s(q)(t(q)) > γ_j(t(q))
            do q := q-1;
            if q < 1 then q := 1; s(1) := j;
            else
            w := 1 + y_s(γ_s(q),γ_j);
                if w < N+1 then q := q+1; s(q) := j; t(q) := w;
                endif
            endif
        endfor

        for j := N to 1 do begin   (backward scan)
            t(x,j) := γ_s(q)(j)
            if j := t(q) then q := q-1
            endif
        endfor
    endfor
end
```

Partial pseudo-code for second step of SEDT.

Coursework

30.1. [possible lab project] Implement the 2D EDT (or the 2D SEDT; it is up to your preference) for binary pictures of your choice. (The size of the processed pictures has to be at least 128×128.)

Note that the lecture notes only provide pseudo code for the second step of the SEDT, which represents the column scans.

Before implementing the algorithm, write the missing part of the pseudo code for the SEDT (for the row scans) and test it manually on the provided example on Page 5.

Visualize results of the distance transform by mapping different distance values (within a defined interval) onto uniquely assigned RGB color values (i.e., this is an example of *pseudo coloring*).

Run your algorithm on pictures of varying size and report the run times in a diagram (Note: for a more accurate measurement, run the algorithm on the same picture n times, and divide total time then by n.) Your diagram should indicate the linear run time of the algorithm.

30.2. Replace the used metric by the Minkowski metrics $d_1((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|$ or $d_\infty((x_1, y_1), (x_2, y_2)) = \max\{|x_1 - x_2| + |y_1 - y_2\}$. Discuss the resulting values of the distance transform by doing (manually) small examples of 2D binary pictures. Can you simplify the second step (the column scan)?

Appendix: 3D and Beyond

The pseudo-code describes the algorithm for the second step, also called *column scan*. The result of row scans (see f_1 and f_2) are represented by γ_j at values $t(j)$. Assume an nD binary picture, $n \geq 2$. After initialization in one (!) direction, computations for every further direction (dimension) can be done independently. The result of calculating a minimum distance for one dimension is an integer value (for each grid point) which will be used for the computation in the next dimension. Index i does not depend on j in the 2D distance transform

$$\begin{aligned} t(x,y) &= \min\{(x-i)^2 + (y-j)^2 : i = 1,...,M \wedge j = 1,...,N\} \\ &= \min\{\min\{(x-i)^2 : i = 1,...,M\} + (y-j)^2 : j = 1,...,N\} \end{aligned}$$

The minimum calculation $g(j) = \min\{(x-i)^2 : i = 1,...,M\}$ corresponds to the row scans in the first step of the algorithm. We can rewrite the equation for fixed x:

$$t(x,y) = \min\{g(j) + (y-j)^2 : j = 1,...,N\}$$

Let p be a 3D location (x,y,k), for $k = 1,...,N_2$, associated with $f_3(x,y,k) = h(k)$, for fixed x,y. Then we use

$$t(x,y,z) = \min\{h(k) + (z-k)^2 : k = 1,...,N_2\}$$

This can continue for arbitrary dimensions $n \geq 4$.