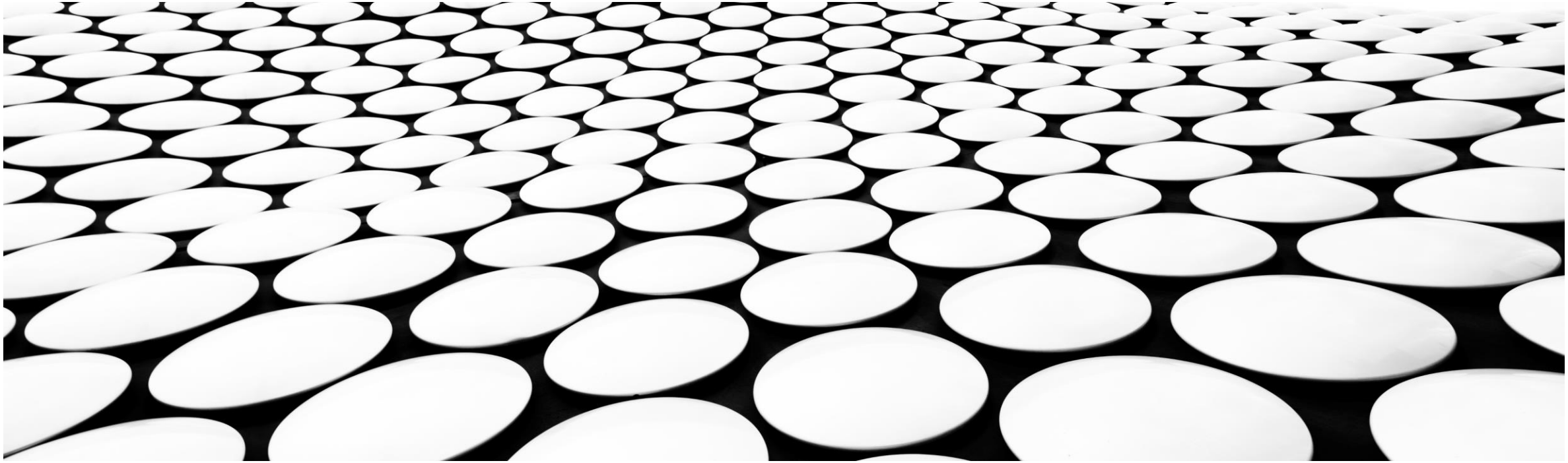
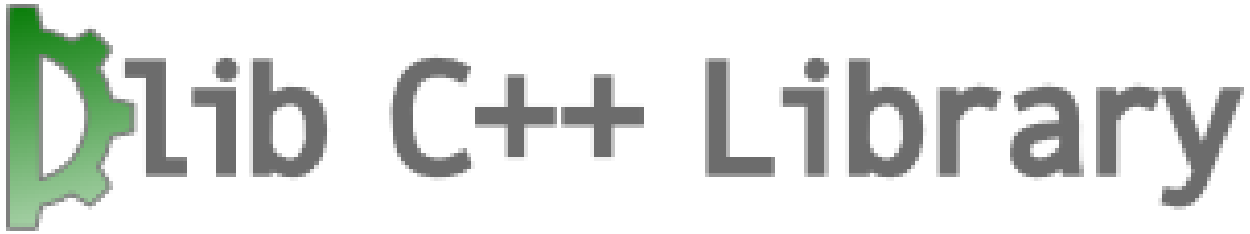
dLib and OpenCV

COEN-317: Distributed Systems
Robert Bruce
Department of Computer Science and Engineering
Santa Clara University



What is dlib?

- The dlib logo [1]:



- dlib is an open source machine learning library [2].
- The dlib application programmer interface is written in C++ [2].
- The dlib library runs on Windows, Mac OS, GNU/Linux, as well as POSIX-compliant operating systems [2].
- We will be using the dlib library to determine the location of sixty-eight Cartesian coordinate facial data points.

[1] http://dlib.net/dlib-logo.png
[2] http://dlib.net/

What is OpenCV?

- The OpenCV logo [1]:

- An open source Computer vision library comprised of over 2500 algorithms [2].
- The OpenCV library has application programmer interfaces for C++, Java, Matlab, and Python [2].
- The OpenCV library runs on Windows, Mac OS, GNU/Linux, and Android [2].
- We will be using OpenCV to compute, then draw Delaunay triangles given a series of sixty-eight facial data points from dlib.

[1] https://opencv.org/resources/media-kit/
[2] https://opencv.org/about/

Creating a face mesh with Delaunay triangles

Creating a face mesh via Delaunay triangles is a multi-step process:

1. The program, *face_landmark_detection* uses dlib to determine the location of the sixty-eight facial data points (Cartesian coordinates) in an input image if there is a human face in the image.

2. The program, *draw_delaunay_triangles* parses the output from *face_landmark_detection*. If there are sixty-eight facial data points, these values are stored in an OpenCV two-dimensional floating-point vector.

3. The program, *draw_delaunay_triangles* uses OpenCV to calculate the Delaunay triangles based on the sixty-eight facial data points.

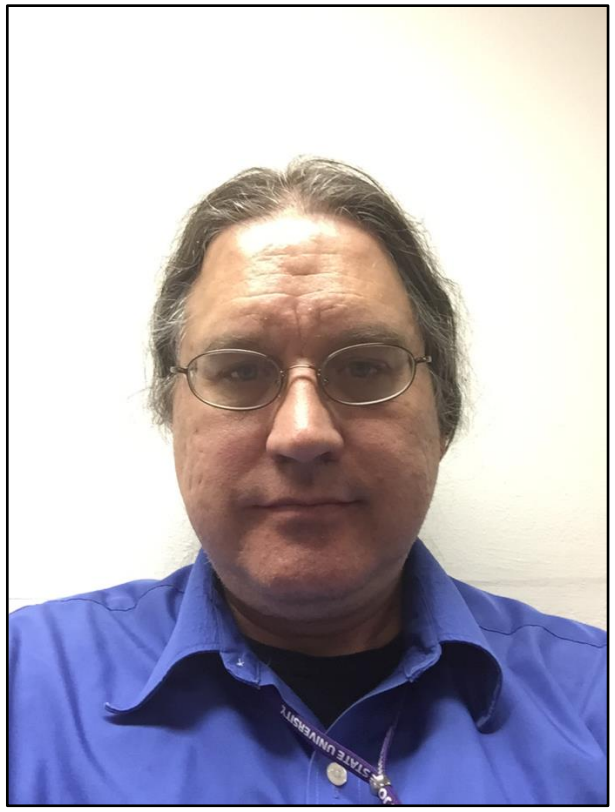
4. The program, *draw_delaunay_triangles* uses OpenCV to draw the Delaunay triangles calculated in step 3 on a copy of the input image. Note: the original input image is not overwritten but a new output file is created instead with a different filename.

Detecting facial data points with dlib

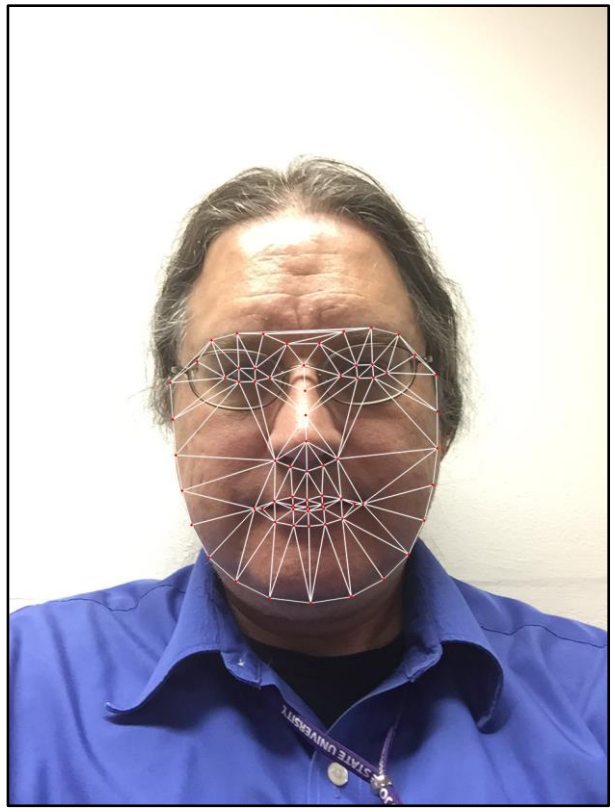
```
face_landmark_detection rob_bruce.png

Face detected in "rob_bruce.png" (258, 603), (262, 662), (267, 719), (277, 776), (292, 831),
(320, 883), (363, 921), (417, 948), (483, 956), (548, 946), (600, 918), (640, 878), (665, 824),
(679, 767), (686, 708), (686, 649), (684, 590), (301, 569), (324, 535), (365, 524), (407, 525),
(445, 541), (497, 540), (536, 521), (580, 515), (624, 526), (650, 559), (472, 575), (474, 616),
(475, 655), (476, 697), (426, 728), (451, 738), (477, 745), (503, 736), (526, 725), (346, 595),
(368, 580), (394, 580), (418, 595), (394, 601), (368, 602), (531, 592), (556, 575), (582, 574),
(605, 587), (584, 595), (558, 596), (393, 805), (425, 793), (454, 786), (477, 791), (501, 784),
(532, 789), (567, 797), (534, 821), (506, 830), (480, 833), (456, 832), (426, 826), (406, 805),
(455, 800), (478, 802), (502, 798), (553, 798), (504, 805), (479, 809), (455, 806)
```

dlib and OpenCV: a visual example

```
draw_delaunay_triangles rob_bruce.png
```

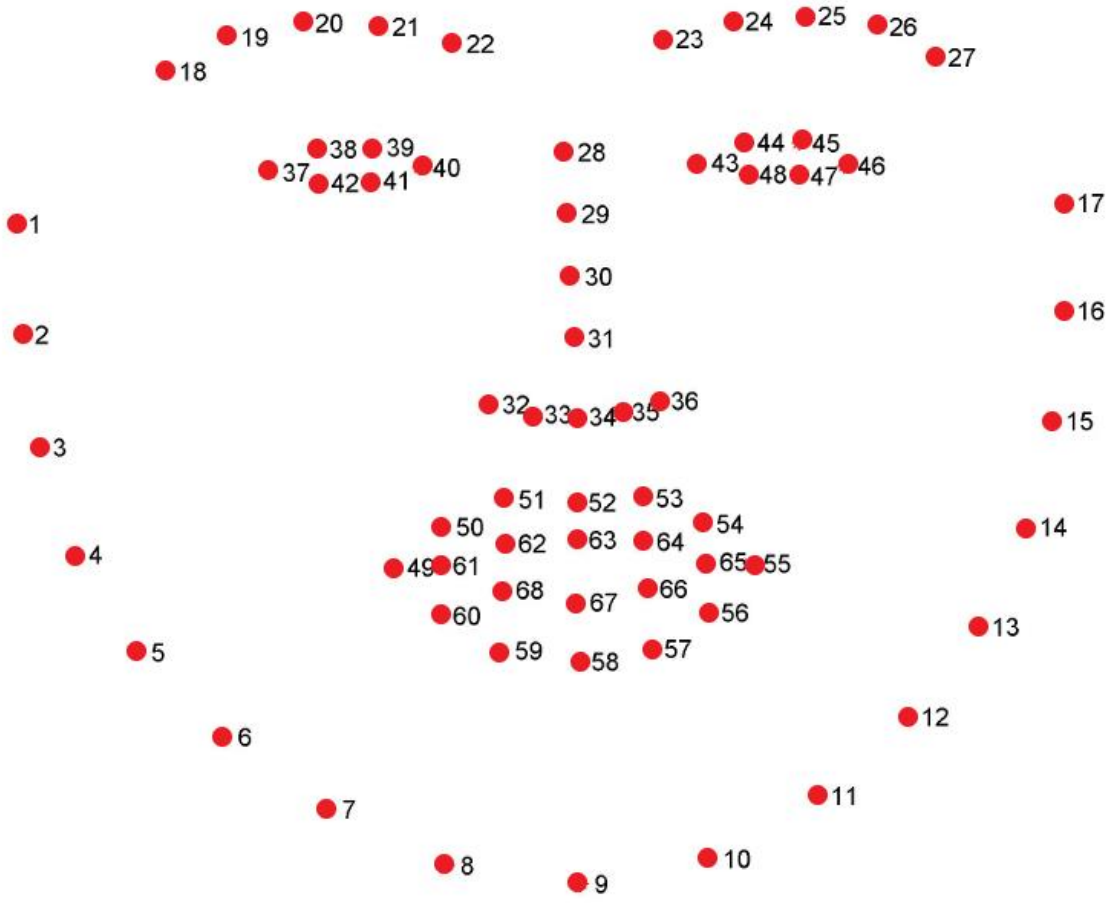


rob_bruce.png

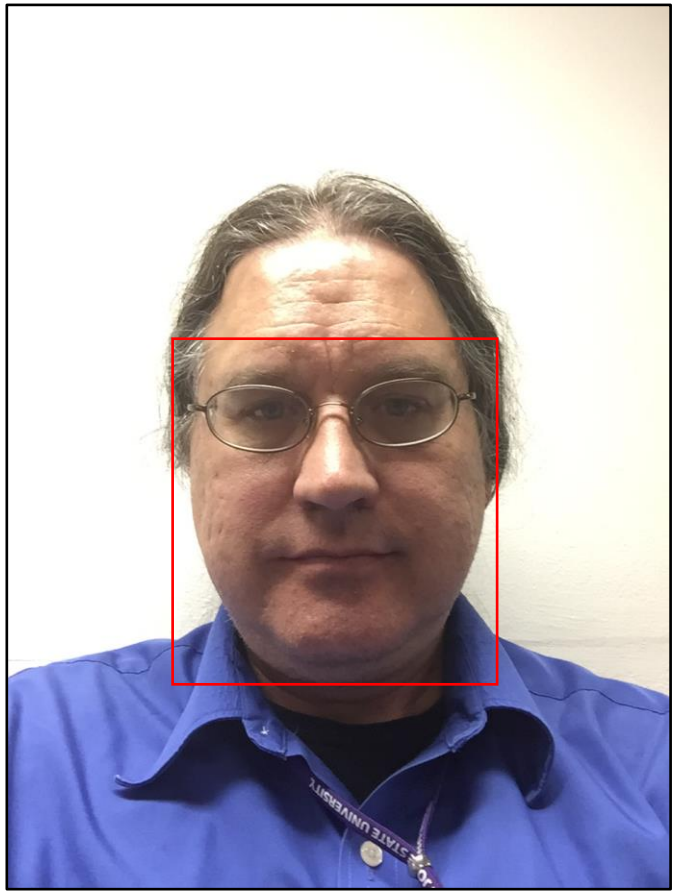


MESH-rob_bruce.png

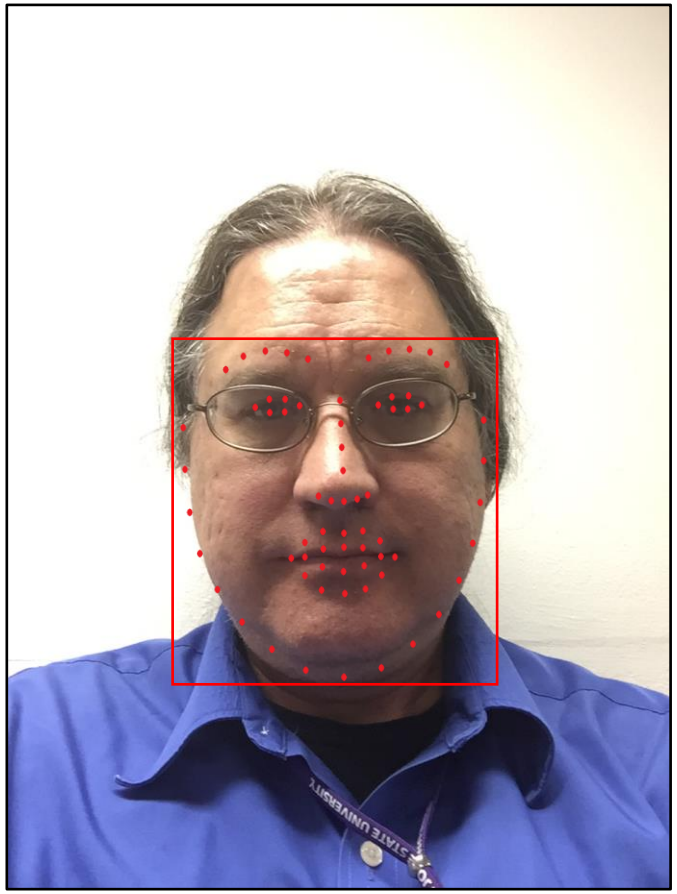
Active Shape Modelling: a template



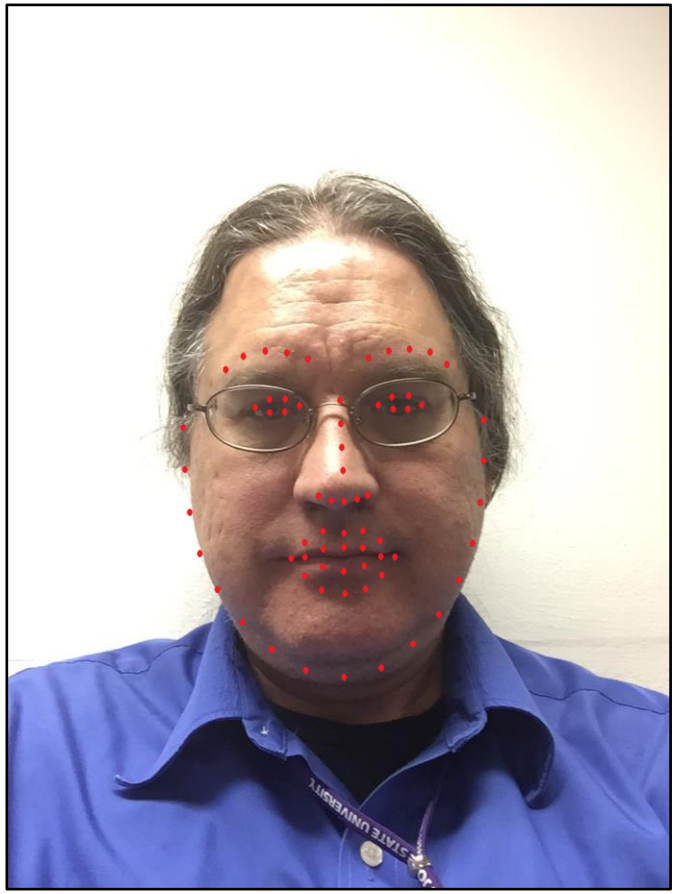
Active Shape Modelling: aligning the template



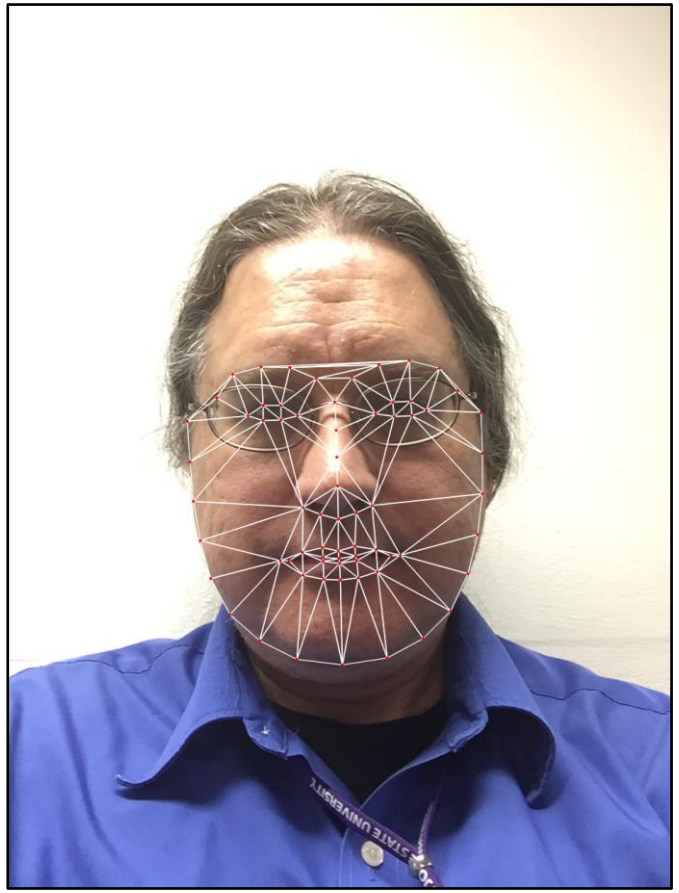
Active Shape Modelling: aligning the template



Active Shape Modelling: aligning the template



Active Shape Modelling: template alignment completed



Dissecting face_landmark_detection.cpp

```
frontal_face_detector detector = get_frontal_face_detector();
shape_predictor sp;
deserialize ("./shape_predictor_68_face_landmarks.dat") >> sp;
if (argc >= 2)
{
  array2d<rgb_pixel> img;
  load_image (img, argv[1]);
  std::vector<rectangle> dets = detector (img);
  if (dets.size() > 0)
  {
    std::vector<full_object_detection> shapes;
    full_object_detection shape = sp(img, dets[0]);
    if (shape.num_parts() == 68)
    {
      printf ("Face detected in \"%s\" ", argv[1]);
      for (unsigned long face_data_point = 0; face_data_point < shape.num_parts() - 1; face_data_point++)
      {
        printf ("(%ld, %ld), ", shape.part(face_data_point).x(), shape.part(face_data_point).y());
      }
      printf ("(%ld, %ld)\n", shape.part(67).x(), shape.part(67).y());
    }
  }
  else
  {
    printf ("No Face detected in \"%s\"\n", argv[1]);
  }
}
```

Dissecting face_landmark_detection.cpp

```
frontal_face_detector detector = get_frontal_face_detector();
shape_predictor sp;
deserialize ("./shape_predictor_68_face_landmarks.dat") >> sp;
if (argc >= 2)
{
  array2d
  load_im
  std::ve
  if (det
  {
    std::vector<full_object_detection> shapes;
    full_object_detection shape = sp(img, dets[0]);
    if (shape.num_parts() == 68)
    {
      printf ("Face detected in \"%s\" ", argv[1]);
      for (unsigned long face_data_point = 0; face_data_point < shape.num_parts() - 1; face_data_point++)
      {
        printf ("(%ld, %ld), ", shape.part(face_data_point).x(), shape.part(face_data_point).y());
      }
      printf ("(%ld, %ld)\n", shape.part(67).x(), shape.part(67).y());
    }
  }
  else
  {
    printf ("No Face detected in \"%s\"\n", argv[1]);
  }
}
```

We begin with standard initializations for face detection in dlib. In the first line, we will ultimately be determining a bounding box for the input image face. That bounding box's length and width will affect the shape of the shape model (i.e. squash and stretch a 68-data point shape image as shown in slide 7).

Dissecting face_landmark_detection.cpp

```
frontal_face_detector detector = get_frontal_face_detector();
shape_predictor sp;
deserialize ("./shape_predictor_68_face_landmarks.dat") >> sp;
if (argc >= 2)
{
  array2d<rgb_pixel> img;
  load_image (img, argv[1]);
  std::vector<rectangle> dets = detector (img);
  if (de
  {
    std
    full
    if (shape.num_parts() == 68)
    {
      printf ("Face detected in \"%s\" ", argv[1]);
      for (unsigned long face_data_point = 0; face_data_point < shape.num_parts() - 1; face_data_point++)
      {
        printf ("(%ld, %ld), ", shape.part(face_data_point).x(), shape.part(face_data_point).y());
      }
      printf ("(%ld, %ld)\n", shape.part(67).x(), shape.part(67).y());
    }
  }
  else
  {
    printf ("No Face detected in \"%s\"\n", argv[1]);
  }
}
```

Load the image file specified on the command line through argv into memory as a two-dimensional array.

Dissecting face_landmark_detection.cpp

```
frontal_face_detector detector = get_frontal_face_detector();
shape_predictor sp;
deserialize ("./shape_predictor_68_face_landmarks.dat") >> sp;
if (argc >= 2)
{
  array2d<rgb_pixel> img;
  load_image (img, argv[1]);
  std::vector<rectangle> dets = detector (img);
  if (dets.size() > 0)
  {
    std::vector<full_object_detection> shapes;
    full
    if
    {
      printf ("Face detected in \"%s\" ", argv[1]);
      for (unsigned long face_data_point = 0; face_data_point < shape.num_parts() - 1; face_data_point++)
      {
        printf ("(%ld, %ld), ", shape.part(face_data_point).x(), shape.part(face_data_point).y());
      }
      printf ("(%ld, %ld)\n", shape.part(67).x(), shape.part(67).y());
    }
  }
  else
  {
    printf ("No Face detected in \"%s\"\n", argv[1]);
  }
}
```

Determine the number of faces detected in the input image. The faces are detected in rectangular bounding box. If the number of bounding boxes is greater than zero, it indicates we found a valid human face!

Dissecting face_landmark_detection.cpp

```
frontal_face_detector detector = get_frontal_face_detector();
shape_predictor sp;
deserialize ("./shape_predictor_68_face_landmarks.dat") >> sp;
if (argc >= 2)
{
  array2d<rgb_pixel> img;
  load_image (img, argv[1]);
  std::vector<rectangle> dets = detector (img);
  if (dets.size() > 0)
  {
    std::vector<full_object_detection> shapes;
    full_object_detection shape = sp(img, dets[0]);
    if (shape.num_parts() == 68)
    {
      p
      f
      {
      }
      p
    }
  }
  else
  {
    printf ("No Face detected in \"%s\"\n", argv[1]);
  }
}
```

With a bounding box containing a human face inside, we can then apply the generic active shape model from slide 7. The active shape model is squashed and stretched to match the contour of the human face detected in the image. The contours of various portions of the active shape model also follow contours of the nose, eyes, eyebrows, chin, upper and lower lips, etc.

Dissecting face_landmark_detection.cpp

```
frontal_face_detector detector = get_frontal_face_detector();
shape_predictor sp;
deserialize ("./shape_predictor_68_face_landmarks.dat") >> sp;
if (argc >= 2)
{
  array2d<rgb_pixel> img;
  load_image (img, argv[1]);
  std::vector<rectangle> dets = detector (img);
  if (dets.size() > 0)
  {
    std::vector<full_object_detection> shapes;
    full_object_detection shape = sp(img, dets[0]);
    if (shape.num_parts() == 68)
    {
      printf ("Face detected in \"%s\" ", argv[1]);
      for (unsigned long face_data_point = 0; face_data_point < shape.num_parts() - 1; face_data_point++)
      {
        printf ("(%ld, %ld), ", shape.part(face_data_point).x(), shape.part(face_data_point).y());
      }
      printf ("(%ld, %ld)\n", shape.part(67).x(), shape.part(67).y());
    }
  }
  else
  {
    pri
  }
}
```

The number of shape parts returned should be sixty-eight. This is because the active shape model uses sixty-eight facial data points (Cartesian coordinates) as a template around a face. Next we output these sixty-eight facial data points to the standard output as Cartesian coordinates. Using standard output in this way decouples the dlib library and internal data structures from other programs in your pipeline.

Dissecting draw_delaunay_triangles.cpp

```
int main (int argc, char *argv[])
{
  char  line[2000], master_buffer[10000],
        input_filename_and_path[MAX_FILENAME_PATH_LENGTH + MAX_FILENAME_LENGTH + 1],
        output_filename_path[MAX_FILENAME_PATH_LENGTH + 1], output_filename[MAX_FILENAME_LENGTH + 1],
        mesh_output_filename_and_path[MAX_FILENAME_PATH_LENGTH + MAX_FILENAME_LENGTH + 1];
  int   i, j, fd[2], n, process_status, child_pid, bytes_read, total_bytes_read,
        x_coordinates[68], y_coordinates[68];
  pid_t wpid;

  total_bytes_read = 0;
  bzero (&master_buffer[0], sizeof(master_buffer));
  bzero (&input_filename_and_path[0], sizeof(input_filename_and_path));
  bzero (&output_filename_path[0], sizeof(output_filename_path));
  bzero (&output_filename[0], sizeof(output_filename));
  if (argc == 2)
  {
    strcpy (&input_filename_and_path[0], argv[1]);
    extract_path_and_filename (&input_filename_and_path[0], sizeof(input_filename_and_path),
                               &output_filename_path[0], sizeof(output_filename_path), &output_filename[0],
                               sizeof(output_filename));
     snprintf (&mesh_output_filename_and_path[0], sizeof(mesh_output_filename_and_path),
               "%sMESH-%s", &output_filename_path[0], &output_filename[0]);
  }
  else
  {
    printf ("Usage: %s <input image filename>\n", argv[0]);
    exit (EXIT_SUCCESS);
  }
```

Dissecting draw_delaunay_triangles.cpp

extract_path_and_filename parses the input path/filename from command line via argv. Example:

input_filename_and_path = "./rob_bruce.png"

After calling extract_path_and_filename:

output_filename_path = "./"
output_filename = "rob_bruce.png"

```
if (argc == 2)
{
  strcpy (&input_filename_and_path[0], argv[1]);
  extract_path_and_filename (&input_filename_and_path[0], sizeof(input_filename_and_path),
                             &output_filename_path[0], sizeof(output_filename_path), &output_filename[0],
                             sizeof(output_filename));
```

Dissecting draw_delaunay_triangles.cpp

```
if (pipe(fd) < 0)
{
  printf ("pipe() failed.\n");
  exit (EXIT_FAILURE);
}
switch (child_pid = fork())
{
  case -1:
    printf ("fork() error\n");
    exit (EXIT_FAILURE);
  case 0:  // child
    dup2 (fd[1], 1);     // 0 is STDIN (read), 1 is STDOUT (write)!
    dup2 (STDOUT_FILENO, STDERR_FILENO); // redirect standard error to standard out.
    close (fd[0]);       // the child does not need this end of the pipe
    execl ("./face_landmark_detection", "./face_landmark_detection", &input_filename_and_path[0], (char *) NULL);
    exit (EXIT_FAILURE); // should not make it here!
  default:               // parent
    close (fd[1]);       // the parent does not need this end of the pipe
    memset (&line[0], 0, sizeof(line));
    bytes_read = read(fd[0], &line[0], sizeof(line) - 1);
    while (bytes_read > 0)
    {
      line[bytes_read] = '\0';
      total_bytes_read = total_bytes_read + bytes_read;
      if (total_bytes_read < (sizeof(master_buffer) - 1))
      {
        strcat (master_buffer, line);
      }
      else
      {
        printf ("Exceeded master buffer size\n");
      }
      bytes_read = read(fd[0], line, sizeof(line) - 1);
    }
    wpid = waitpid (child_pid, &process_status, 0);
    if (wpid < 0)
    {
      printf ("waitpid error\n");
      exit (EXIT_FAILURE);
    }
  break;
}
```

Dissecting draw_delaunay_triangles.cpp

```
if (pipe(fd) < 0)
{
  printf ("pipe() failed.\n");
  exit (EXIT_FAILURE);
}
switch (child_pid = fork())
{
  case -1:
    printf ("fork() error\n");
    exit (EXIT_FAILURE);
  case 0:  // child
    dup2 (fd[1], 1);     // 0 is STDIN (read), 1 is STDOUT (write)!
    dup2 (STDOUT_FILENO, STDERR_FILENO); // redirect standard error to standard out.
    close (fd[0]);       // the child does not need this end of the pipe
    execl ("./face_landmark_detection", "./face_landmark_detection", &input_filename_and_path[0], (char *) NULL);
    exit (EXIT_FAILURE); // should not make it here!
  default:               // parent
    close (fd[1]);       // the parent does not need this end of the pipe
    memset (&line[0], 0, sizeof(line));
    bytes_read = read(fd[0], &line[0], sizeof(line) - 1);
    while (bytes_read > 0)
    {
      line[bytes_read] = '\0';
      total_bytes_read = total_bytes_read + bytes_read;
      if (total_bytes_read < (sizeof(master_buffer) - 1))
      {
        strcat (master_buffer, line);
      }
      els
      {
        pi
      }
      byt
    }
    wpid =
    if (wp
    {
      pri
      exi
    }
  break;
}
```

Notice how I use the familiar fork() command to launch a child process. The child process executes the "face_landmark_detection program (to detect a face). That program then outputs the sixty-eight facial data points (as Cartesian coordinates) to standard output.

Since standard output for the child process is being piped to the parent process, the parent process will read then store the coordinates in an input buffer. The parent process will need to parse the input buffer to extract each (X,Y) cartesian coordinate.

Dissecting draw_delaunay_triangles.cpp

```
  // ************************************
  // * Initialize the coordinates in array *
  // ************************************
  for (j = 0; j < NUM_DLIB_FACIAL_COORDINATES; j++)
  {
    x_coordinates[j] = 0;
    y_coordinates[j] = 0;
  }
  if (extract_cartesian_coordinates_from_input_buffer (&master_buffer[0], total_bytes_read, &x_coordinates[0], &y_coordinates[0]))
  {
    draw_face_mesh (&input_filename_and_path[0], &mesh_output_filename_and_path[0], &x_coordinates[0], &y_coordinates[0]);
  }
  else
  {
    printf ("draw_face_mesh() failed to properly parse output from face_landmark_detection program\n");
  }
  exit (EXIT_SUCCESS);
}
```

Dissecting draw_delaunay_triangles.cpp

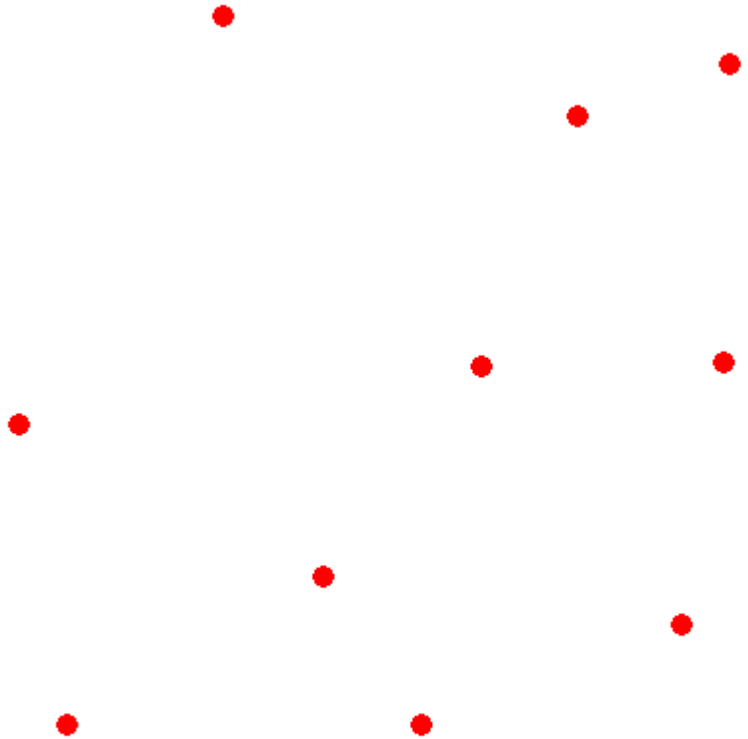
```
  // ************************************
  // * Initialize the coordinates in array *
  // ************************************
  for (j = 0; j < NUM_DLIB_FACIAL_COORDINATES; j++)
  {
    x_coordinates[j] = 0;
    y_coordinates[j] = 0;
  }
  if (extract_cartesian_coordinates_from_input_buffer (&master_buffer[0], total_bytes_read, &x_coordinates[0], &y_coordinates[0]))
  {
    draw_face_mesh (&input_filename_and_path[0], &mesh_output_filename_and_path[0], &x_coordinates[0], &y_coordinates[0]);
  }
  else
  {
    printf ("draw_face_mesh() failed to properly parse output from face_landmark_detection program\n");
  }
  exit (EXIT_SUCCESS);
}
```

The subroutine responsible for parsing the output from child process to parent process is: extract_cartesian_coordinates_from_input_buffer()

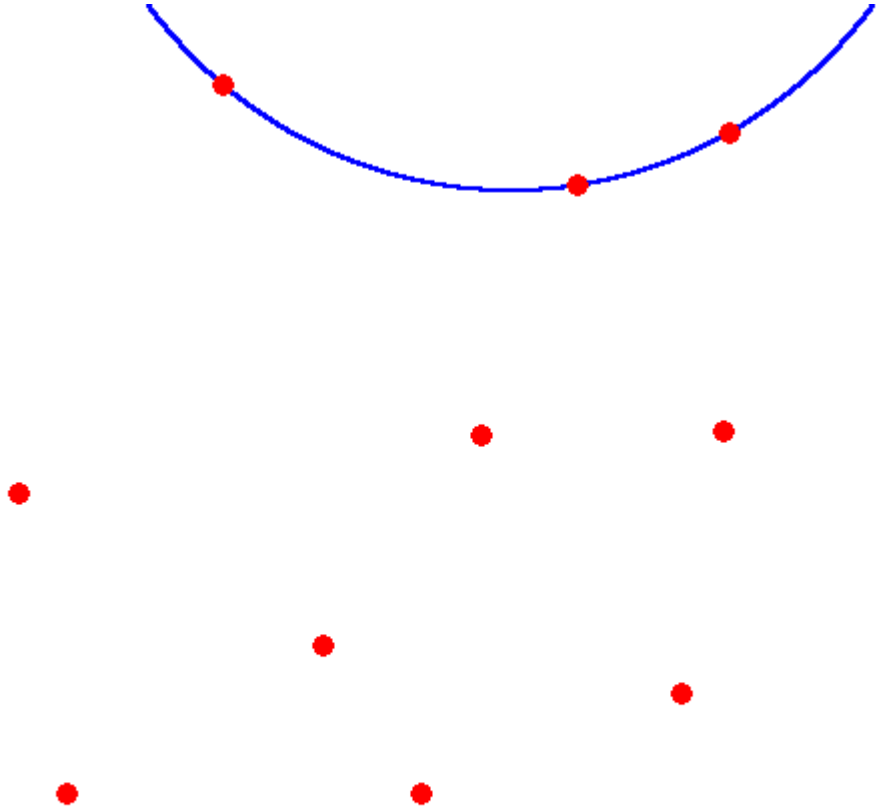
If successful, that subroutine returns sixty-eight Cartesian coordinates into two arrays: An X array and a Y array. For example, X[0] and Y[0] store the first of the sixty-eight facial data points.

Lastly, we call draw_face_mesh with the input image filename and path, the output image filename and path, and the X, and Y coordinates.

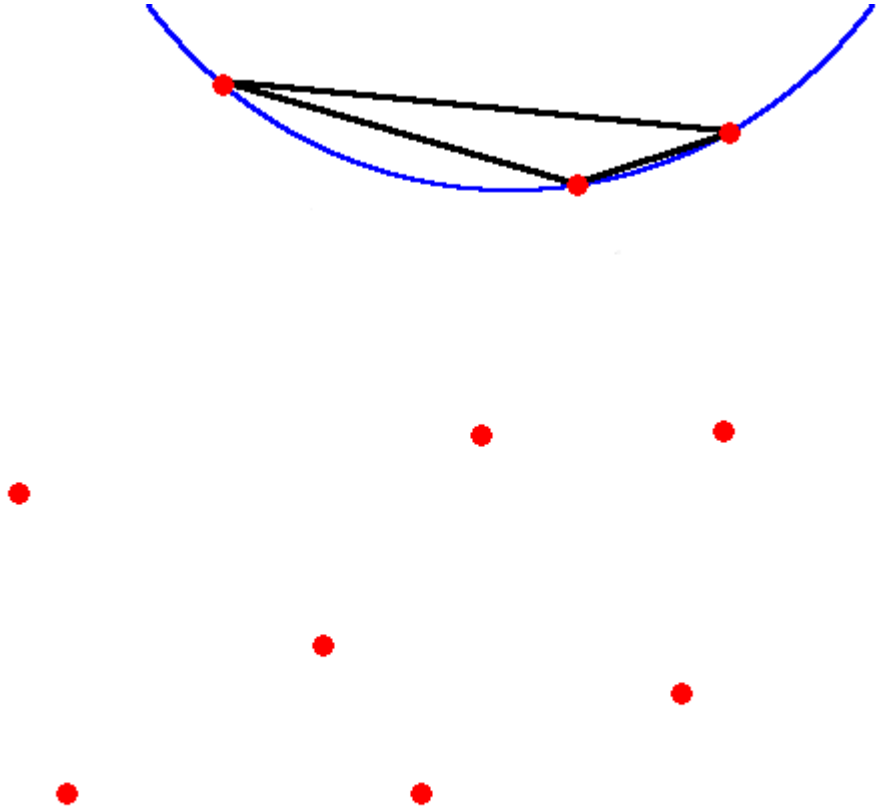
Computing Delaunay triangles: a visual example



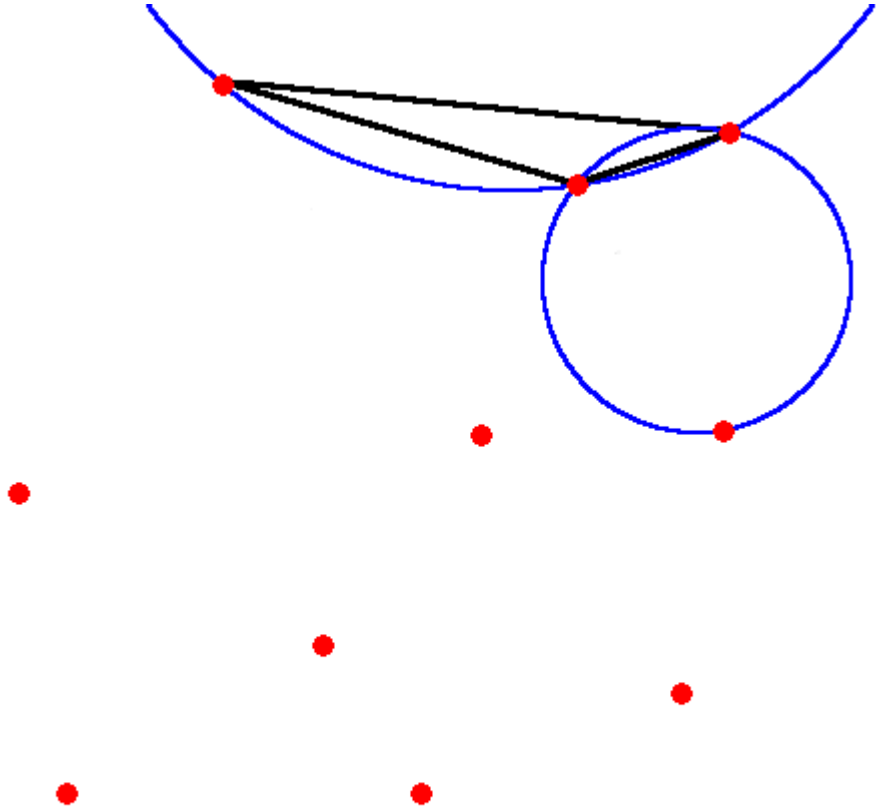
Computing Delaunay triangles: a visual example



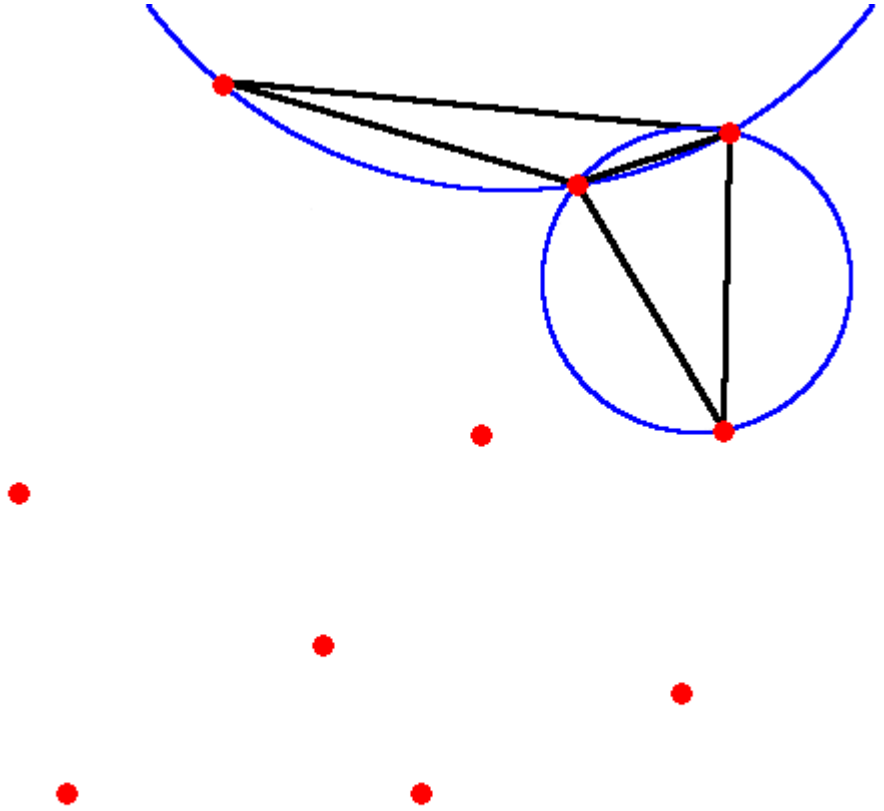
Computing Delaunay triangles: a visual example



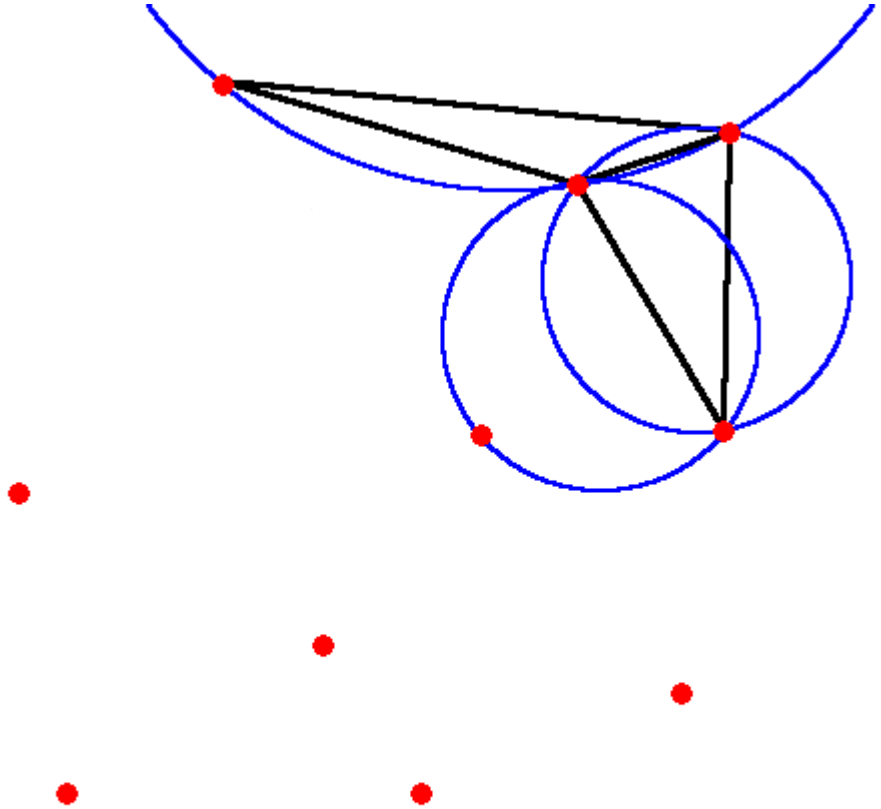
Computing Delaunay triangles: a visual example



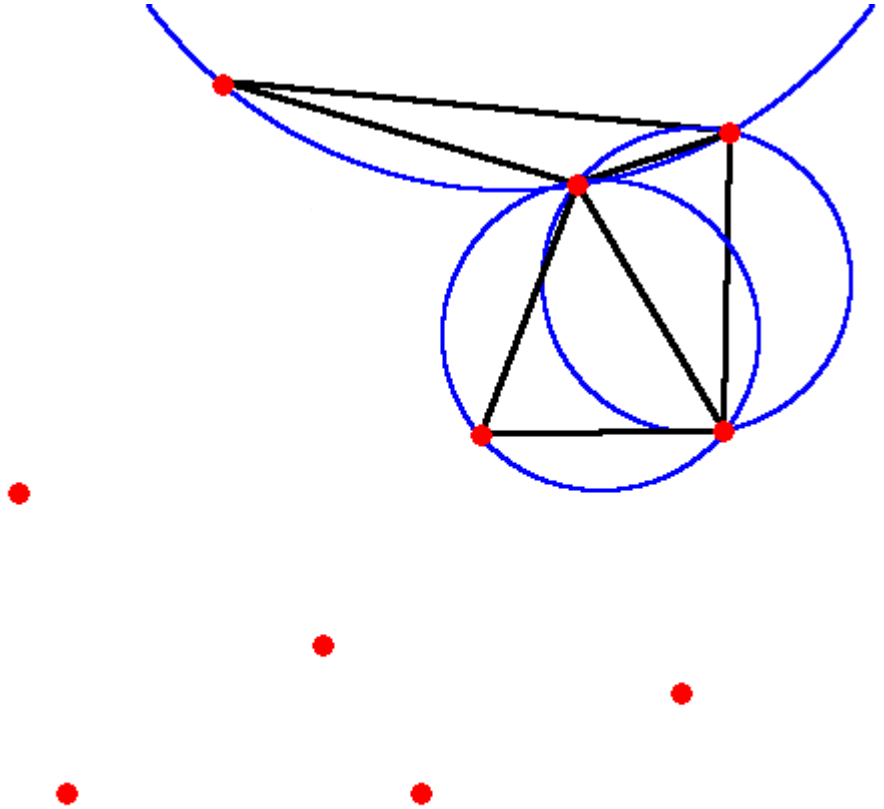
Computing Delaunay triangles: a visual example



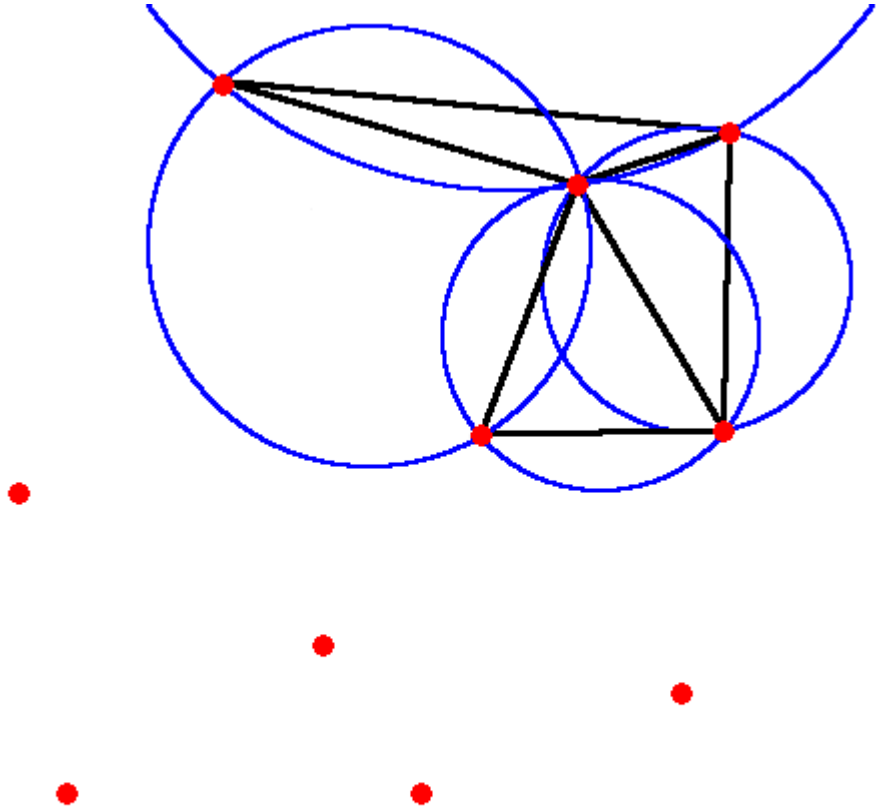
Computing Delaunay triangles: a visual example



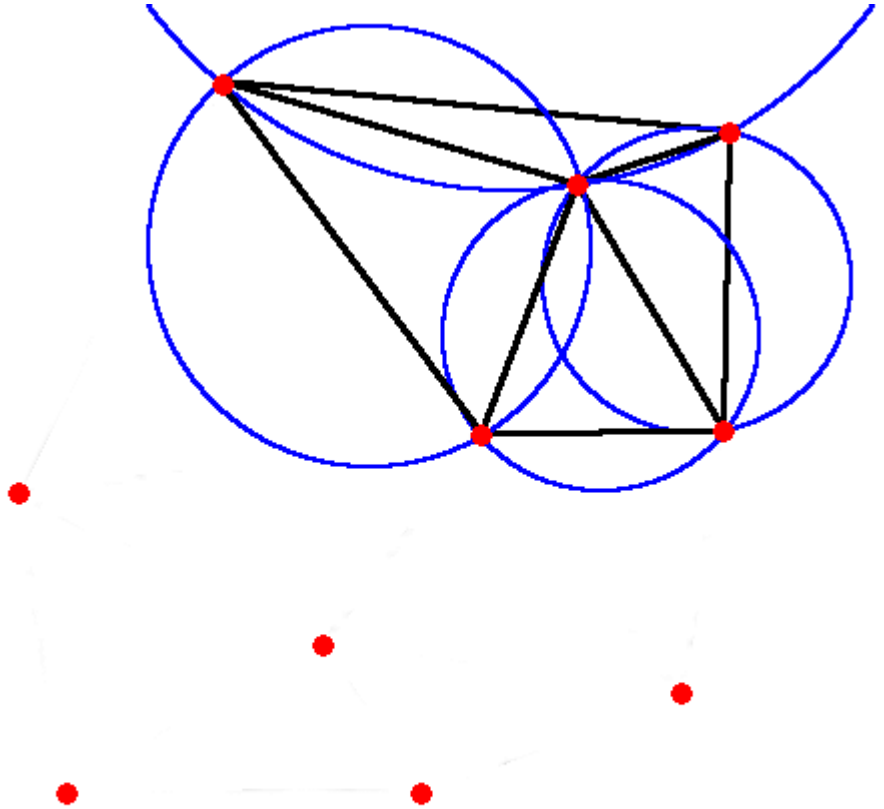
Computing Delaunay triangles: a visual example



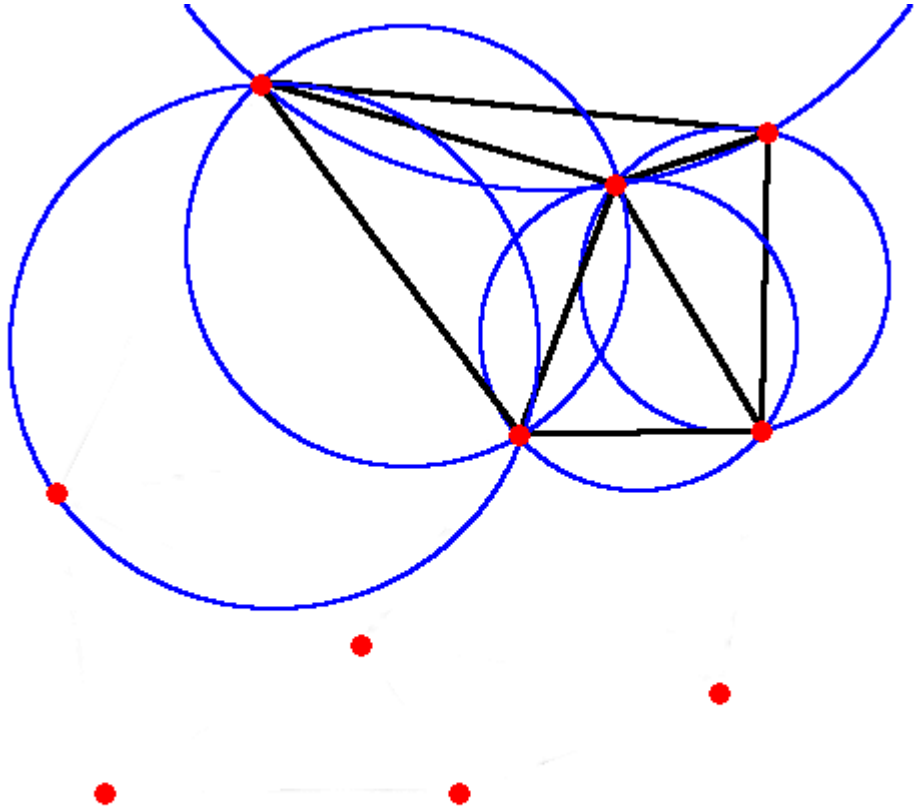
Computing Delaunay triangles: a visual example



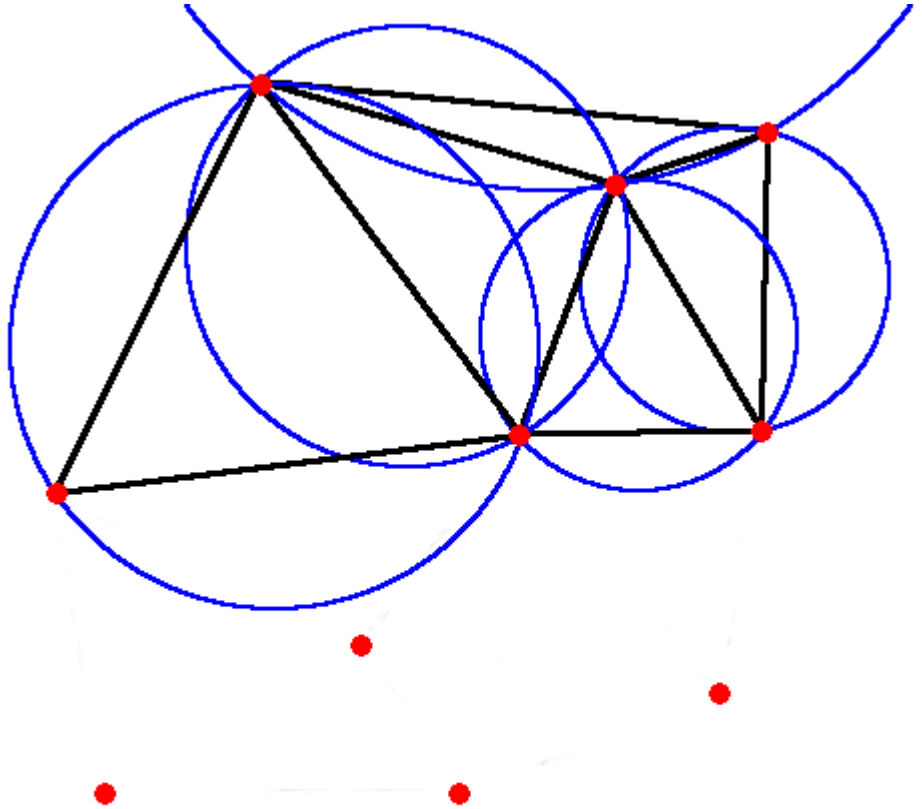
Computing Delaunay triangles: a visual example



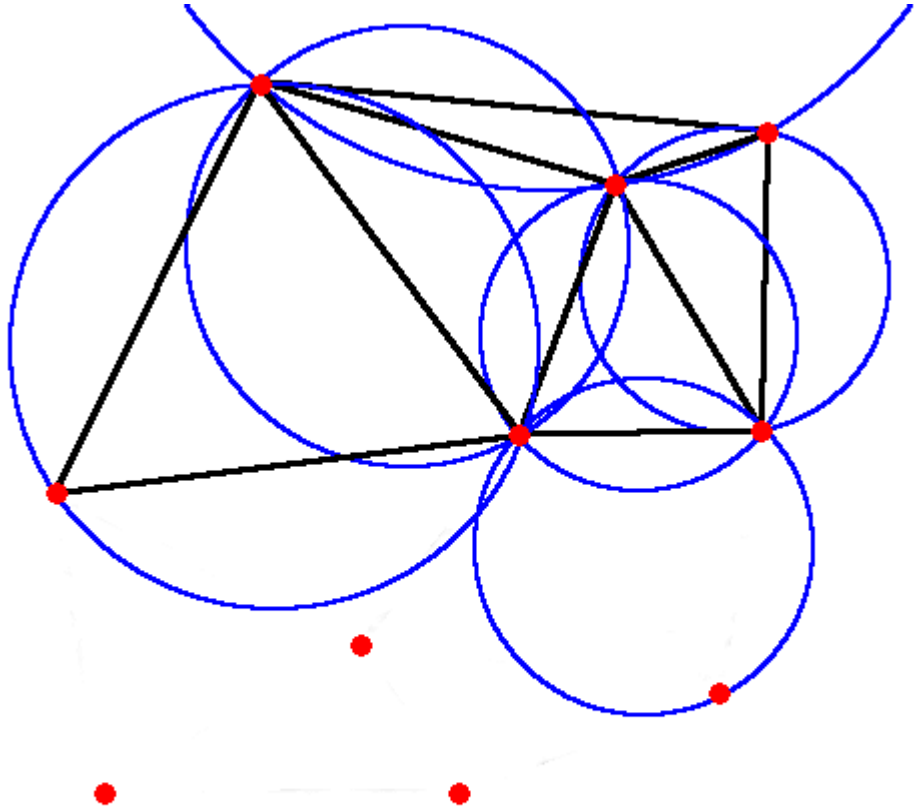
Computing Delaunay triangles: a visual example



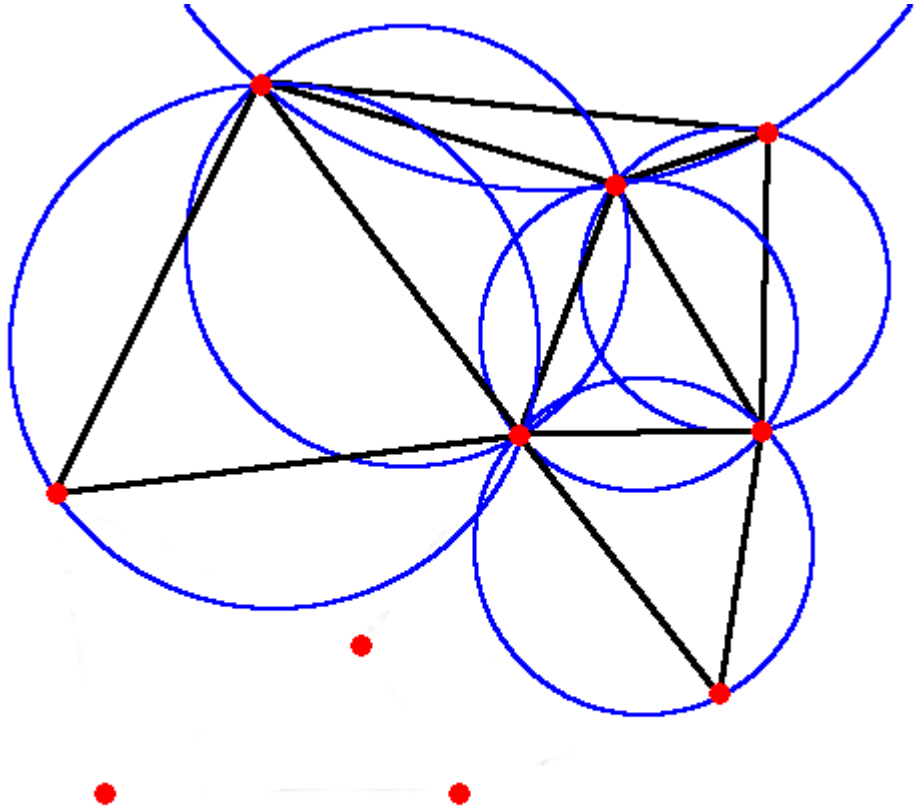
Computing Delaunay triangles: a visual example



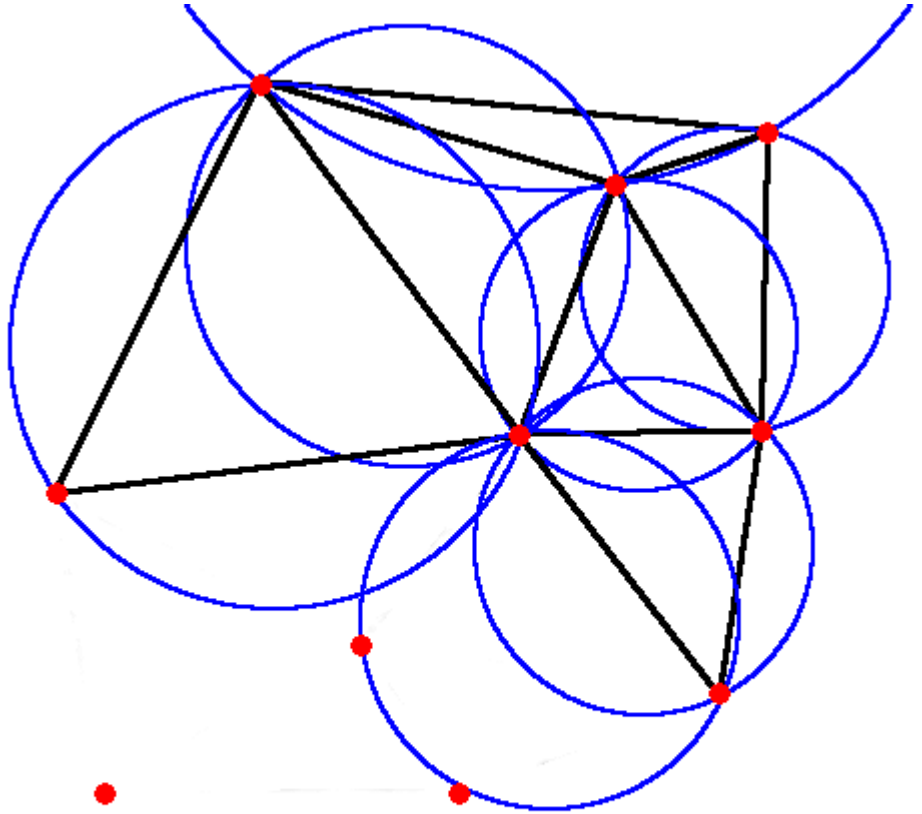
Computing Delaunay triangles: a visual example



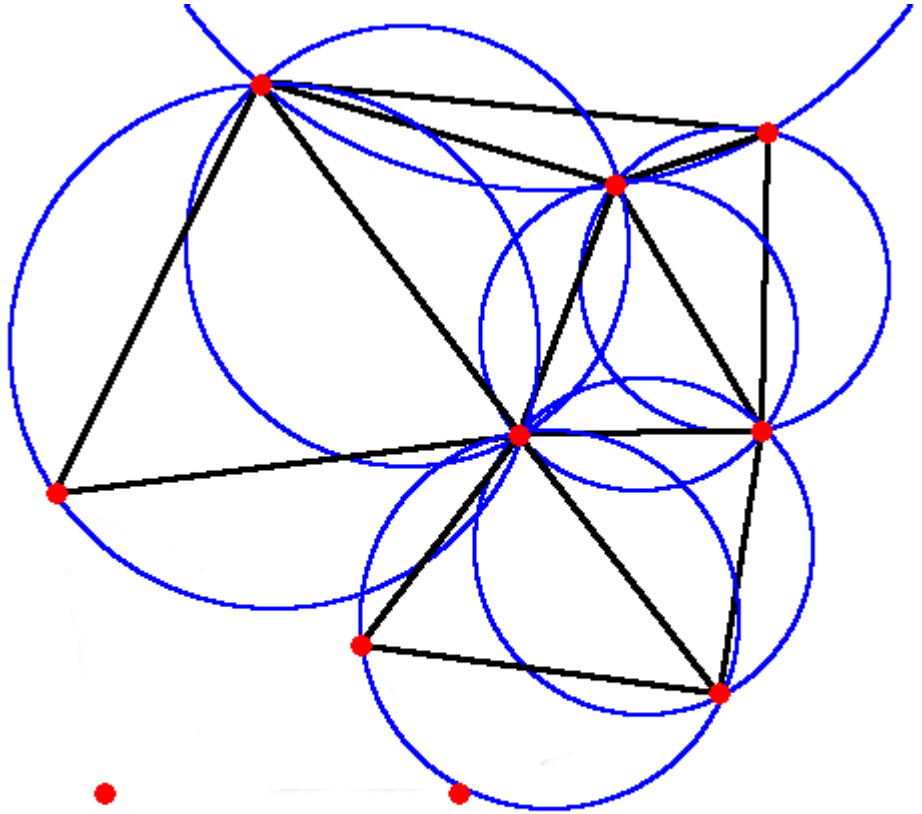
Computing Delaunay triangles: a visual example



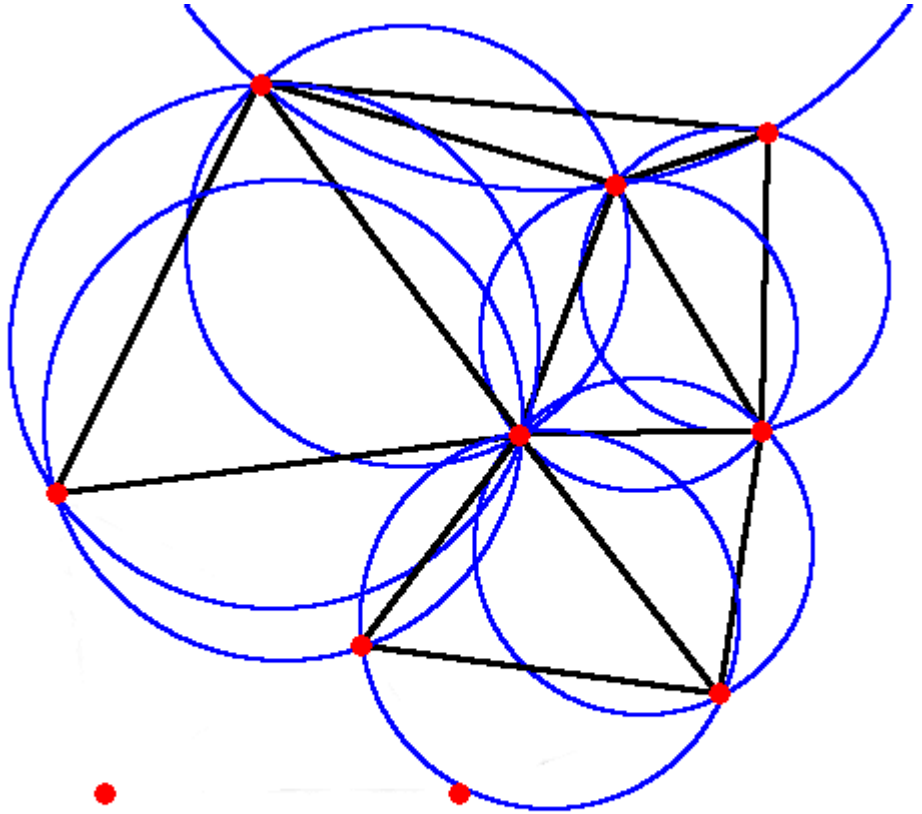
Computing Delaunay triangles: a visual example



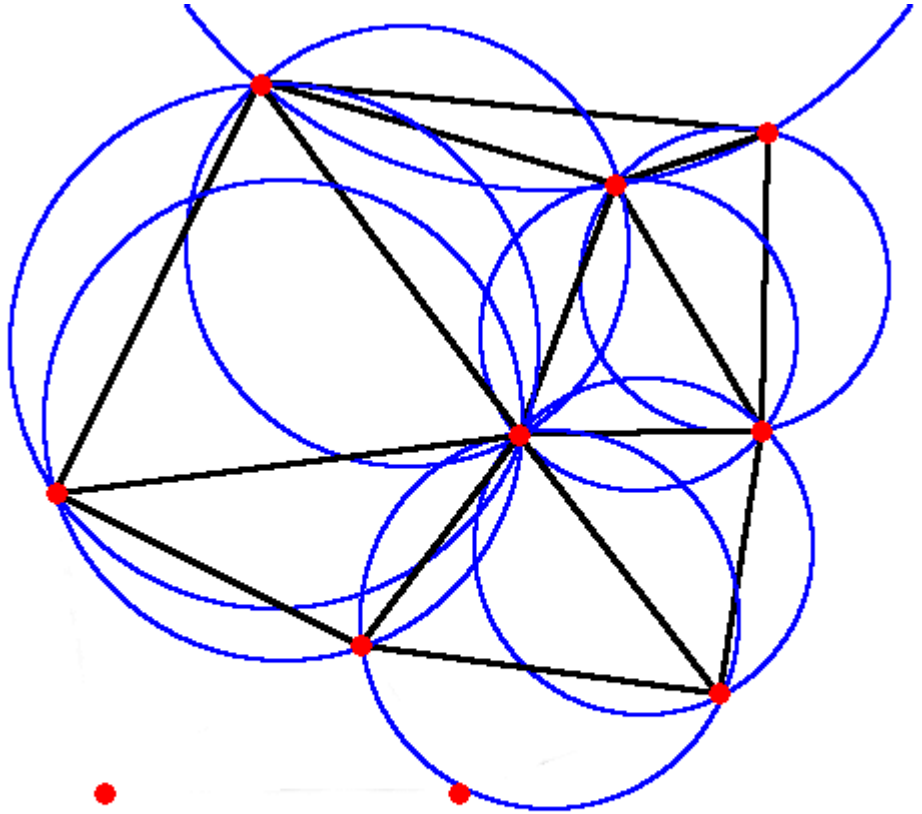
Computing Delaunay triangles: a visual example



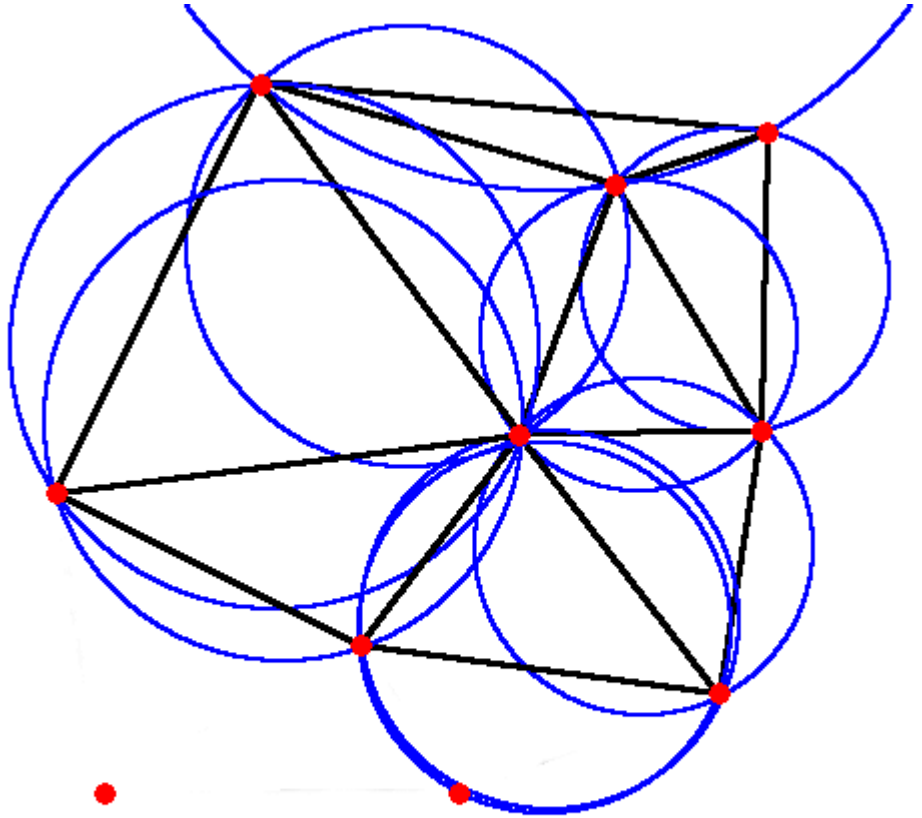
Computing Delaunay triangles: a visual example



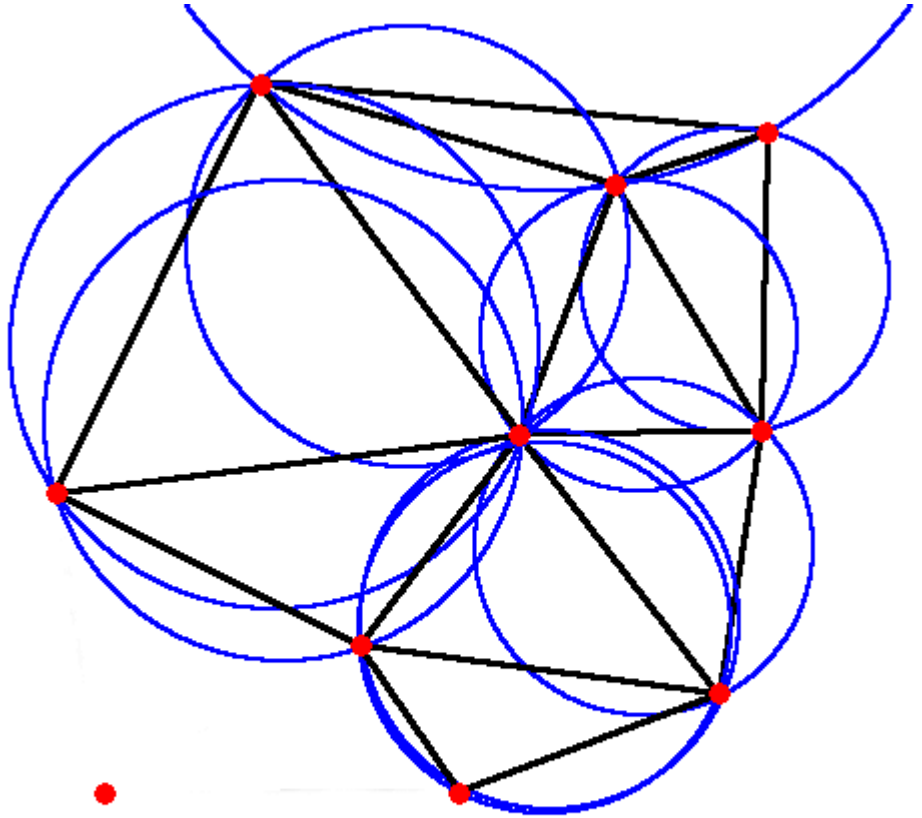
Computing Delaunay triangles: a visual example



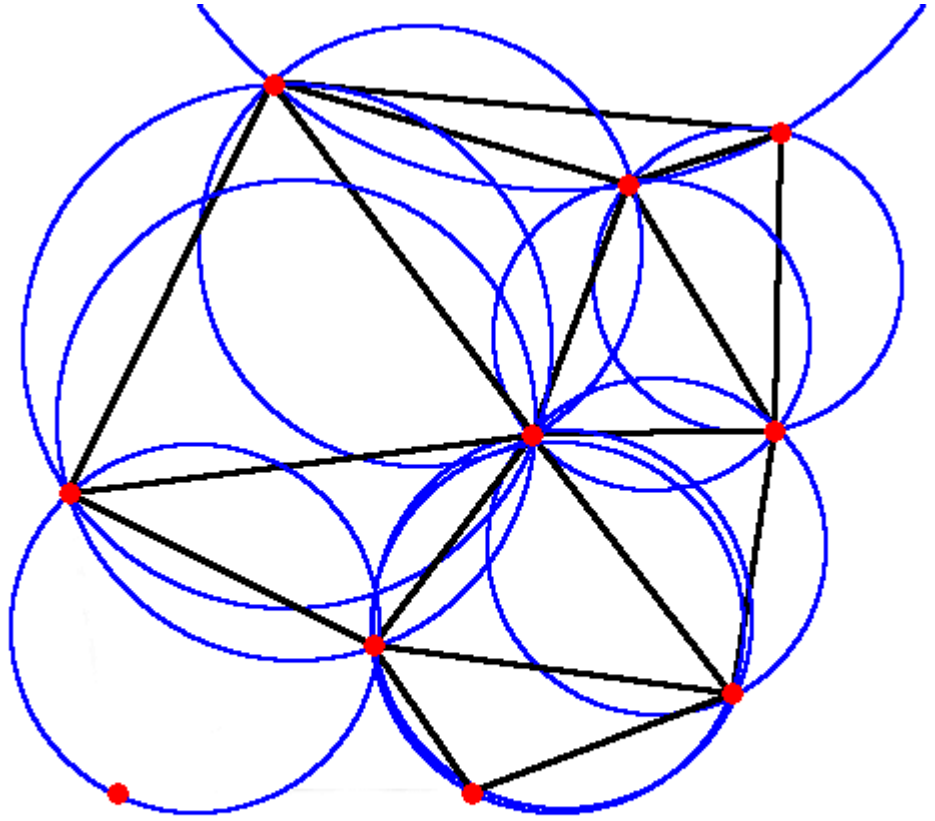
Computing Delaunay triangles: a visual example



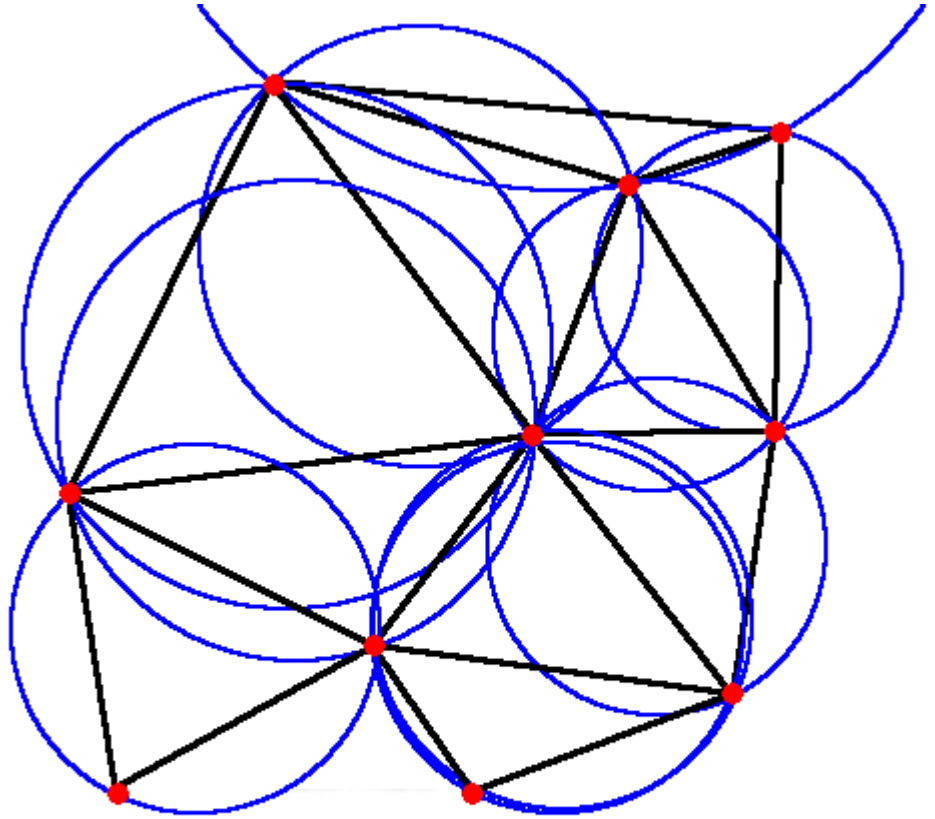
Computing Delaunay triangles: a visual example



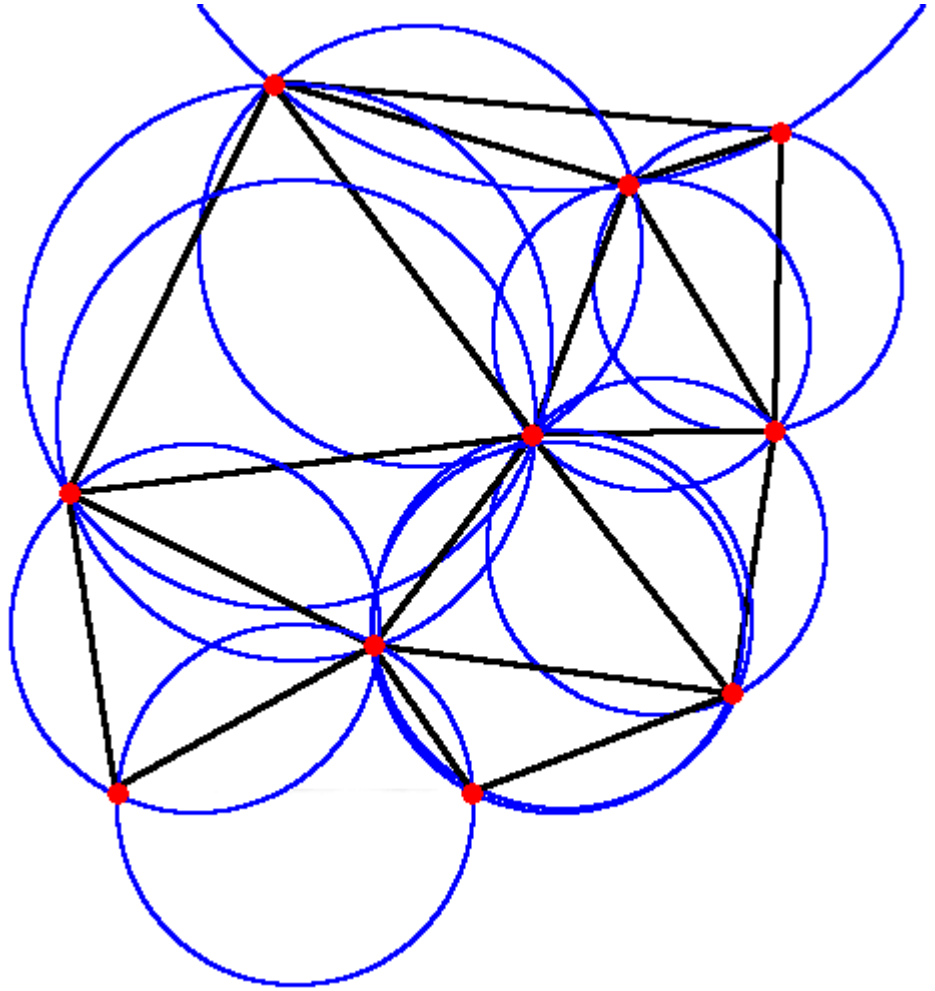
Computing Delaunay triangles: a visual example



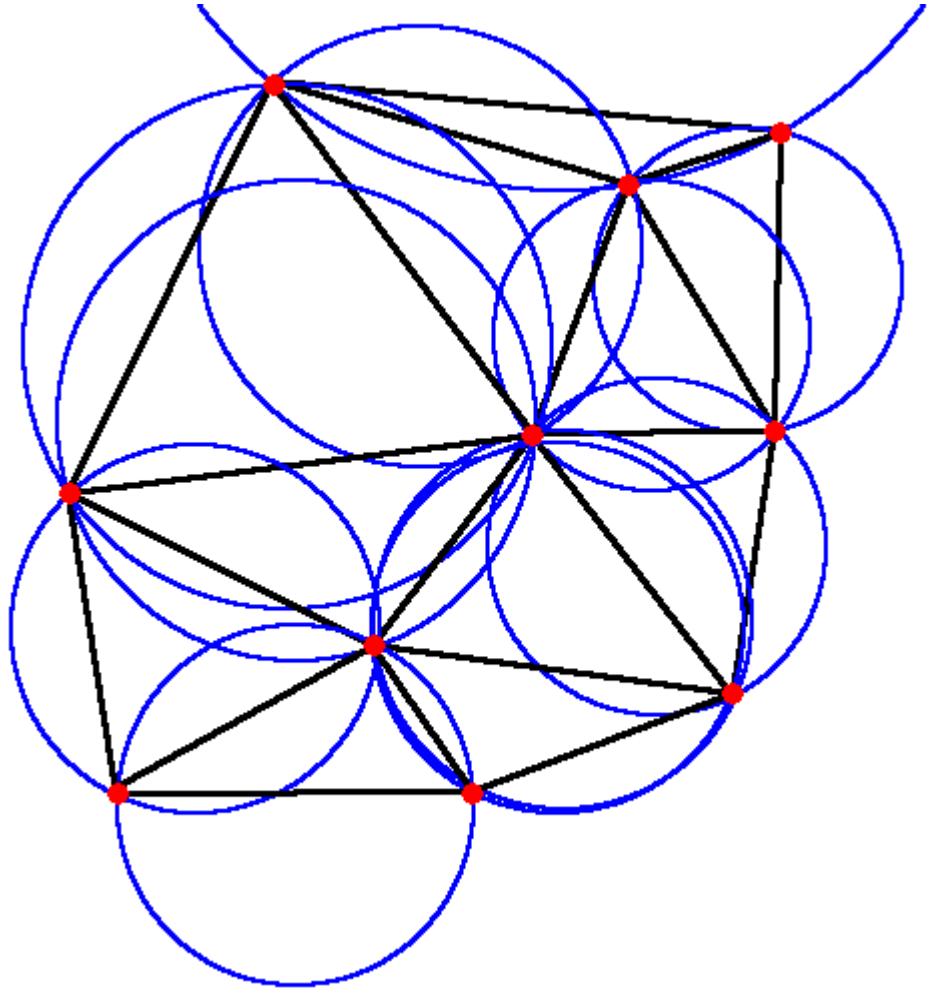
Computing Delaunay triangles: a visual example



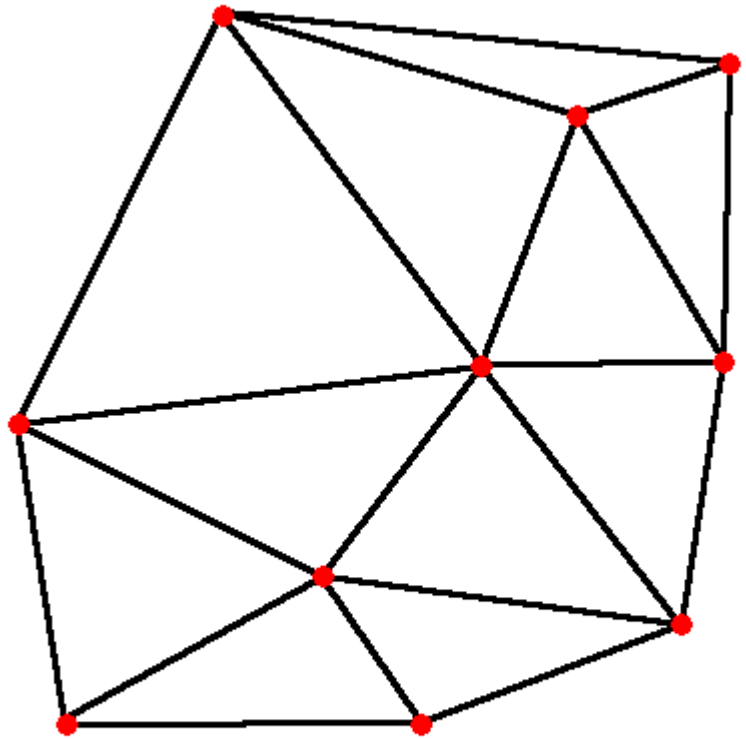
Computing Delaunay triangles: a visual example



Computing Delaunay triangles: a visual example



Computing Delaunay triangles: a visual example



For further reading...

- Example dlib program to detect faces. This program is available on Camino under Files: Example programs:

  face_landmark_detection.tar.gz

- Example OpenCV program to draw Delaunay triangles. This program is available on Camino under Files: Example programs: draw_delaunay_triangles:

  draw_delaunay_triangles.c

- dLib source code and documentation:

  http://dlib.net/

- OpenCV source code and documentation:

  https://opencv.org/