SDL2$_{g}fxutils -$ $documentationDocumentation$

Release latest

May 19, 2017

Contents

Contents:

CHAPTER 1

SDL2_gfxutils presentation

library **SDL2_gfxutils**

version 2.0.0

platform Linux, Windows, (Posix Compliant not tested).

compiler gcc, g++, clang, mingw32, mingw-w64-686, mingw-w64-x86_64.

synopsis **SDL2_gfxutils** a **SDL2_gfx** forms generating and manipulating helper functions set

Licence GPLv3

author Eddie Brüggemann <mrcyberfighter@gmail.com>

SDL2_gfxutils brief history

SDL2_gfxutils is issue from an collection of functions that i have implemented over the years, everytime i had a idea of a form to generate i try to implemented it as a function generating the sequence of coordinates or performing operations on a form. At start i implemented my ideas with the *python pygame* module, which is based on the **SDL** library, by start i get only the functions to set a pixel according the a radius and an angle offset, so **SDL2_gfxutils** is a translation of *python* functions into the **C** language and the **SDL2_gfx** standart.

SDL2_gfxutils has been entirely rewritten after the first version release which suffers under severals bugs and was not handy to use, mostly for animations.

note After the disaster everything are right.

The new implementation from SDL2_gfxutils

All forms generating functions now return a **pointer** on a specific **SDL2_gfxutils** type compatible with the generic **SDL2_gfxutils** *Form* type.

The **pointers** permit to **manipulate** the forms for **transforming** or **animating** (*rotating*, *scaling*, *translating*, *mirroring*) functions **easily**.

The **pointers** can be destroyed at your convienence of course. And this mechanic is massively used in form generating functions. So that no memory space is lost.

The generic *Form* type has been change to contains coordinates from type `float`, instead of type `int16_t`, so that the **computation** like *rotating*, *scaling*, *translating*, *mirroring* are now exactly executed.

The subtype `Pixel` members are now from type `float`.

- Some functions have been removed because they become useless, because of the new pointers system.
- Some functions have been added for replacing the missing features, with many advantages, mostly for transforming or animating forms in the SDL2 mainloop.
- All functions have been improved, favor of the pointers mechanic.

SDL2_gfxutils presentation

SDL2_gfxutils is en extension for the **SDL2_gfx** library helping you for the creation of the fantastics drawing your brain can imagined.

SDL2_gfxutils provide severals functions for severals usages:

- A lot of forms generating functions, from the simple **polygons**, through **stars**, to the fantastic **fractals**.
- High-level Transforming or animating functions (*rotating*, *scaling*, *translating* and *mirroring*) acting on an entire *Form*.
- Low-level Transforming or animating functions (*rotating*, *scaling*, *translating* and *mirroring*) acting on a single *Pixel*.
- Memory management and check functions.

  note *I think it's easy to adapt the SDL2_gfxutils library to be compatible with others libraries than the SDL2 library*.

Operations functions which transform a form:

- **Rotation** of a *Form* around his center from the wanted **degrees**.
- **Scaling** of a *Form* from the wanted **factor**.
- **Translation** of a *Form* from the wanted `x` and `y` values.
- **Mirroring** over the `X` or `Y` **axes** according to an `center` **point**.

Form setters functions which change the settings of a form

- Setting a new `center` of a *Form* with optionally translating all coordinates.
- Setting a new `radius` what equal to scaling a *Form* except that instead a factor you can set a new size directly.
- Setting a new `color` for the *Form* or a *Line*.

Form getters functions to get settings of a form

- Getting the current *Form* `center` value.
- Getting the current *Form* `color`.

- Getting the current *Form* `length` (often the **radius**).
- Getting the current *Form* `orientation` (**offset** defining the incline of a *Form*).
- Getting the current *Form* `real_length` (value defining the **distance** between the `center` and the **most distant coordinate** from the `center`).

Note: You can use this member to build bounding boxes for collision detection per example.

Displaying forms functions

Each *Form* type has specific displaying functions.

Use each *Form* specific displaying function else the result will be undefined.

note But you can use the displaying function you want to display a *Form*, something the output is surprising.

Note: For every displaying function it exist a *thickness setttable displaying function* and an *anti-aliasing displaying function* except for the filled forms functions.

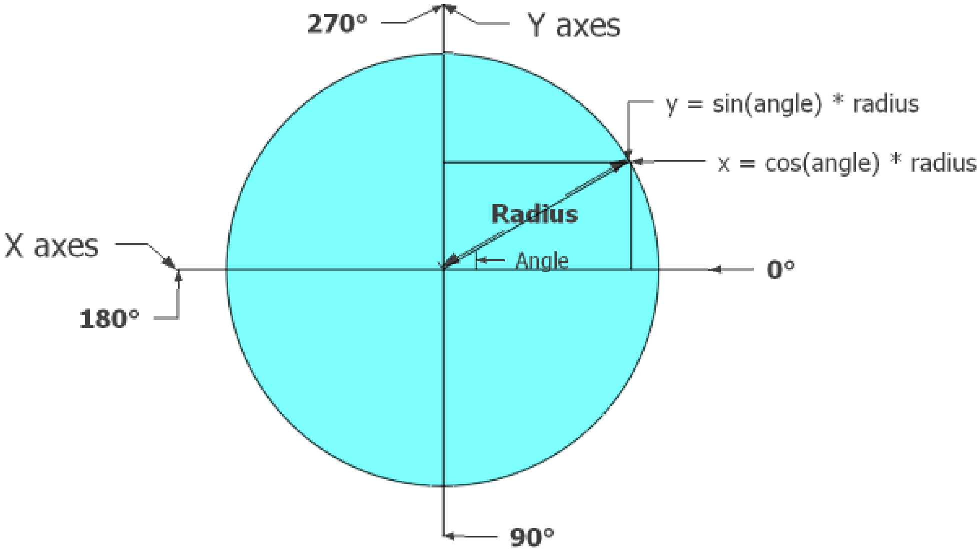
Length and middle Between 2 Pixels

- *Measure of the length of a Line*.
- *Get the Middle Pixel of a Line*.

Angles

In the 2D display from SDL2: the **X axes** goes from left to right and the **Y axes** from up to down.

For the multiple used `orientation` parameter from type `float`. Sea the following image to become acquainted with the values and with the conventional angle values.



note You can sea that 3 o'clock represent 0 degrees.

Animations advice

Polygon and forms are not only displayable object but can also be an guideline for the execution of an animation which moving a form trough the way of the lines from the polygon or the form.

SDL2_gfxutils provide an function *compute_trajectory()* with which you can construct an moving line by moving a form through the pixels of the `positions` array by translating it with the *translate_form()* function.

SDL2_gfxutils Documentation License

This 'SDL2_gfxutils Documentation' is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License.

To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/4.0/>.

SDL2_gfxutils License

Note: SDL2_gfxutils is under copyright from the GNU General Public License.

SDL2_gfxutils a **SDL2_gfx** forms generating and manipulating helper functions set.

Copyright (©) 2016 Brüggemann Eddie <mrcyberfighter@gmail.com>.

SDL2_gfxutils is free software: you can redistribute it and/or modify

it under the terms of the GNU General Public License as published by

the Free Software Foundation, either version 3 of the License, or

(at your option) any later version.

SDL2_gfxutils is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

GNU General Public License for more details.

You should have received a copy of the GNU General Public License

along with **SDL2_gfxutils**. If not, see <http://www.gnu.org/licenses/>

Credits

Thank's

Thank's to my **mother**, **my family** and **the doctors**.

Stay away from drugs: drugs **destroy** your *brain* and your *life*.

CHAPTER 2

SDL2_gfxutils defined types

Following types are define by **SDL2_gfxutils**:

```
typedef struct Color_ {
  uint8_t r ;
  uint8_t g ;
  uint8_t b ;
  uint8_t a ;
} Color ;


typedef struct Pixel_ {
  float x ;
  float y ;
} Pixel ;

typedef struct Segment_ {
  Pixel xy1   ;
  Pixel xy2   ;
  Color color ;
} Line ;

typedef struct Coords_ {
  float *x ;
  float *y ;
} Coords ;

typedef struct Polygon_ {
  Coords         coords      ;
  Pixel          center      ;
  Color          color       ;
  uint16_t       count       ;
  float          length      ;
  float          real_length ;
  float          orientation ;
} Polygon ;
```

```

typedef Polygon Arc          ;
typedef Polygon Hexagram     ;
typedef Polygon Pentagram    ;
typedef Polygon Star         ;
typedef Polygon Spiral       ;


typedef Polygon         Form ;
typedef Arc             Form ;
typedef Hexagram        Form ;
typedef Pentagram       Form ;
typedef Spiral          Form ;
typedef Star            Form ;
```

The Color type

The `Color` is used for holding datas about colors channels:

```
typedef struct Color_ {
  uint8_t r ;
  uint8_t g ;
  uint8_t b ;
  uint8_t a ;
} Color ;
```

The Pixel type

The `Pixel` type is used for holding the datas about an pixel:

> the `x` and `y` coordinate values.

The `Pixel` type is used by functions which return a single pixel.

```
typedef struct Pixel_ {
  float x ;
  float y ;
} Pixel ;
```

Like the function *get_middle_from_line()*.

The Line type

The `Line` type is used for holding the datas about a line:

- The line start `Pixel`.
- The line end `Pixel`.
- The color of the line as **red**, **green**, **blue** and **alpha** values.

The `Line` type is used by functions which return a segment.

```
typedef struct Segment_ {
  Pixel xy1   ;
  Pixel xy2   ;
  Color color ;
} Line ;
```

Like the function *generate_segment()*.

Or as argument from a function per example to get the middle of a line.

The Coords type

The `Coords` type in only used internally to be a member of the `Polygon` type.

```
typedef struct Coords_ {
  float *x ;
  float *y ;
} Coords ;
```

The `Coords` is used for performing computation.

Note: By displaying operations the `float` array members are converted to `int16_t` (the **SDL2_gfx** coordinates arrays standart type for displaying) in this way:

```
int c ;

for (c=0 ; c < form->count-2 ; c++) {

  ret=lineRGBA(pRenderer,
               (int16_t) roundf(form->coords.x[c]),
               (int16_t) roundf(form->coords.y[c]),
               (int16_t) roundf(form->coords.x[c+1]),
               (int16_t) roundf(form->coords.y[c+1]),
               form->color.r,
               form->color.g,
               form->color.b,
               form->color.a) ;
}
```

note After using a Form, Wenn you do not need it in the future, you can free the form.

The Polygon type

The `Polygon` is used for holding all datas about a form.

- The **coordinates** for *computing* the form generation and used by the animation functions: the `Coords struct`.
- The **center** of the form: the `Pixel struct`.
- The **color** of the form: the `r`, `g`, `b`, `a` members.
- The **count** of coordinates number: the `count` member.
- The **length** often the *radius* needed from the animating functions: the `length` member.

- The **length** between the `center` and the most distant **coordinate** from the `center`: the `real_length` member.
- The **offset** defining the incline of a form: the `orientation`.

```
typedef struct Polygon_ {
  Coords          coords      ;
  Pixel           center      ;
  Color           color       ;
  uint16_t        count       ;
  float           length      ;
  float           real_length ;
  float           orientation ;
} Polygon ;
```

note All derived types are define as an *Form* type so that you don't need to cast it if you use a `Form` generic function.

Note: Dynamically settings.

All forms generating functions set the `count`, `length`, `center`, `orientation` and the `real_length` member from the `Polygon` type.

Generic Form type.

There are many forms representing derivated types from the *Polygon* type.

```
typedef Polygon Arc          ;
typedef Polygon Hexagram     ;
typedef Polygon Pentagram    ;
typedef Polygon Star         ;
typedef Polygon Spiral       ;
```

All derivated types are define as a generic `Form` type.

```
typedef Polygon         Form ;
typedef Arc             Form ;
typedef Hexagram        Form ;
typedef Pentagram       Form ;
typedef Spiral          Form ;
typedef Star            Form ;
```

Notice

Note: Compatiblity with others libraries than **SDL2**:

The only purpose of the **SDL2_gfx** library is the form displaying functionality.

So I think it's possible to adapt easily **SDL2_gfxutils** to be resusable with others libraries.

You only have to implement the displaying functions adapted to the target library.

If the coordinates arrays from type `float` does it for the target library, because it’s easy to **round** and **cast** the `float` in the target type, like this:

```
int x = (int) roundf(form->coords.x[c]) ;
int y = (int) roundf(form->coords.y[c]) ;
```

Else the *colors* are coded on `uint8_t` values.

And the other members from the *Form* structure are used for computing.

Thank’s to notify me at <mrcyberfighter@gmail.com> if you want to do so.

CHAPTER 3

Base functions

Here are describe the geometric, base functions functions, from **SDL2_gfxutils** used from the differents forms generating functions.

Angles

float **get_angle** (int *position*, float *scale*, float *orientation*)

Parameters

- **position** (int) – Needed for positional purpose: number of scaling units.
- **scale** (float) – Scaling of the angle.
- **orientation** (float) – An additionnal offset to add.

Return type float

Returns An angle in degrees.

This function return an angle value according to the given settings,

by applying following formel:

```
360.0/scale * position + orientation
```

Distance

float **get_distance_pixels** (Pixel *px1*, Pixel *px2*)

Parameters

- **px1** (float) – The start pixel from the distance.
- **px2** (Pixel) – The end pixel from the distance.

Return type float

Returns The distance between `px1` and `px2`.

Pixel

Pixel **get_pixel_coords** (uint32_t *position*, uint32_t *scale*, float *length*, Pixel *center*, float *orientation*)

Parameters

- **position** (`uint32_t`) – An unsigned integer needed for positional purpose.
- **scale** (`uint32_t`) – Scaling of the angle.
- **length** (`float`) – radius.
- **center** (`Pixel`) – center.
- **orientation** (`float`) – An additionnal offset to add.

Return type *Pixel*

Returns The pixel initialized in relationship to the given settings.

This function return an pixel initialized in relationship to the given settings,

by first getting the angle:

```
float angle_degrees = get_angle(position, scale, orientation)   ;
float radians = angle_degrees / 180.0 * PI  ;
```

and applying the following formel to get the `x` and `y` values:

```
Pixel.x = cosf(radians) * length + center.x ;
Pixel.y = sinf(radians) * length + center.y ;
```

Pixel **get_middle_from_line** (Line *line*)

Parameters

- **line** (`Line`) – The line to get the middle point from.

Return type *Pixel*

Returns The pixel middle point from the given line.

Line

Line ***generate_segment** (Pixel *start_point*, float *length*, float *angle*)

Parameters

- **start_point** (`Pixel`) – The segment start point.
- **length** (`float`) – The length of the segment.
- **angle** (`float`) – segment incline *angle*.

Return type *Line **

Returns An line starting at `start_point` from length `length` incline from *angle*.

Arc

Arc ***generate_circle_arc**(float *radius*, Pixel *center*, float *start_pos*, float *circle_part*)

Parameters

- **radius** (float) – the radius starting from the center argument.
- **center** (Pixel) – The center from where generate the circle arc.
- **start_pos** (float) – The start position given as an *angle in degress*.
- **circle_part** (float) – An angle value, in degrees, representing the part of an entire circle of the arc and so the length of the arc in relationship to the radius.

Return type *Arc **.

Returns

A pointer on an *Arc*.

- from radius radius.
- from center center.
- length from part of an circle circle_part.
- starting at offset start_pos which will give the start point from the arc.

CHAPTER 4

Forms generating functions

Polygons

Polygon ***generate_polygon_radius** (uint32_t *sides*, float *radius*, Pixel *center*, float *orientation*)

Parameters

- **sides** (uint32_t) – The number of sides of the polygon to generate.
- **radius** (float) – The radius of the polygon to generate.
- **center** (Pixel) – The wants center of the polygon.
- **orientation** (float) – An offset in degrees to add to influence the incline of the polygon.

Return type *Polygon **

Returns

A regular convex polygon .

note This result in a regular polygon with `sides` sides with radius length `radius` starting at `orientation`.

see *A blue 12 sides polygon convex*.

note An polygon is convex if all vertex from the polygon are on the same side from every edge of the polygon.

Polygon ***generate_corners_rounded_polygon** (uint32_t *sides*, float *radius*, Pixel *center*, float *orientation*)

Parameters

- **sides** (uint32_t) – The number of sides of the polygon to generate.
- **radius** (float) – The radius of the polygon to generate.
- **center** (Pixel) – The wanted center of the polygon.

- **orientation** (float) – An offset in degrees to add to influence the incline of the polygon.

Return type *Polygon **

Returns

A polygon which corners are arcs which size is in relationship to the number of sides of the polygon.

see *A blue 12 sides rounded corners polygon*.

Note The radius goes from the center to the center of the circle arcs: the rounded corners.

Polygon ***generate_sides_rounded_polygon**(uint32_t *sides*, float *radius*, Pixel *center*, float *orientation*)

Parameters

- **sides** (uint32_t) – The number of sides of the polygon to generate.
- **radius** (float) – The radius of the polygon to generate.
- **center** (Pixel) – The wanted center of the polygon.
- **orientation** (float) – An offset in degrees to add to influence the incline of the polygon.

Return type *Polygon **

Returns

A polygon which sides are rounded according the number of sides of the polygon.

see *A blue 12 sides rounded corners polygon*.

Note The radius goes from the center to the center of the circle arcs.

> **Warning:** The parameter sides must be conform to:
>
> ```
> sides % 2 != 0
> ```
>
> I can only generate odd sides numbered sides rounded polygons.

Polygon ***generate_rounded_inside_out_polygon**(uint32_t *sides*, float *radius*, Pixel *center*, float *orientation*)

Parameters

- **sides** (uint32_t) – The number of arcs of the polygon to generate.
- **radius** (float) – The radius of the polygon to generate.
- **center** (Pixel) – The wanted center of the polygon.
- **orientation** (float) – An offset in degrees to add to influence the incline of the polygon.

Return type *Polygon **

Returns

A polygon alternating inside nad outside arcs the number of sides of the polygon.

see *A blue 12 sides rounded inside out polygon*.

Note The radius goes from the center to the center of the circle arcs.

> **Warning:** The sides number is multiply per 2 to obtains an even numbered polygon.

Polygon ***generate_alternate_inside_half_circle_polygon**(uint32_t *sides*, float *radius*, Pixel *center*, float *orientation*)

Parameters

- **sides** (`uint32_t`) – The number of sides of the polygon to generate.
- **radius** (`float`) – The radius of the polygon to generate.
- **center** (`Pixel`) – The wanted center of the polygon.
- **orientation** (`float`) – An offset in degrees to add to influence the incline of the polygon.

Return type *Polygon **

Returns

A rounded polygon alternating arcs rounded to the outside and to the inside of the polygon.

see *A blue 12 sides alternate inside half circle polygon*.

Note The radius goes from the center to the center of the circle arcs.

> **Warning:** The result is an even polygon of the double of the sides values.

Polygon ***generate_alternate_outside_half_circle_polygon**(uint32_t *sides*, float *radius*, Pixel *center*, float *orientation*, bool *side_arcs*)

Parameters

- **sides** (`uint32_t`) – The number of sides of the polygon to generate.
- **radius** (`float`) – The radius of the polygon to generate.
- **center** (`Pixel`) – The wanted center of the polygon.
- **orientation** (`float`) – An offset in degrees to add to influence the incline of the polygon.
- **sides_arcs** (`bool`) – Boolean value determine if the sides which are not half-cirlce arcs should be rounded.

Return type *Polygon **

Returns

A polygon with half-circle rounded to the inside from the half sum from the sides of the polygon and the other is even an arc or an straight line according to the `side_arcs` boolean value.

see *A blue 12 sides alternate outside half circle polygon*.

Note The radius goes from the center to the center of the circle arcs.

> **Warning:** The result is an even polygon of the double of the sides values.

Stars

Star ***generate_star** (uint32_t *pikes*, float *radius*, Pixel *center*, float *orientation*)

Parameters

- **pikes** (uint32_t) – The number of pikes of the star.
- **radius** (float) – The radius of the star base polygon not the spike.
- **center** (Pixel) – The wants center from the star.
- **orientation** (float) – An offset in degrees to add to influence the incline of the star.

Return type *Star **

Returns

A star with the number of wants pikes according to the given settings.

note This function generate a star based on a regular polygon.

see *A blue 24 peaks star*.

Note The radius value is the radius of the base polygon.

Star ***generate_pentagram_star** (float *radius*, Pixel *center*, float *orientation*)

Parameters

- **radius** (float) – An base value for the generation of the 5 pikes star.
- **center** (Pixel) – The wants center from the 5 pikes star.
- **orientation** (float) – An offset in degrees to add to influence the incline of the star.

Return type *Star **

Returns

A not a regular 5 pikes star but a pentagram star or pentacle.

note This function generate a simply 5 extremity star with the particularity that the resulting star is not a regular star but a pentagram star.

see *A blue pentagram star*.

Note The radius value is the radius of the base polygon.

Star ***generate_hexagram_star** (float *radius*, Pixel *center*, float *orientation*)

Parameters

- **radius** (float) – An base value for the generation of the 6 pikes star.
- **center** (Pixel) – The wants center from the 6 pikes star.
- **orientation** (float) – An offset in degrees to add to influence the incline of the star.

Return type *Star **

Returns

A not a regular 6 pikes star but a hexagram star or star of David.

note This function generate a simply 6 extremity star, with the particularity that the resulting star is not a regular star but an hexagram star or star of David.

see *A blue hexagram star*.

Note The radius value is the radius of the base polygon.

pentagram

Pentagram ***generate_pentagram** (float *radius*, Pixel *center*, float *orientation*)

Parameters

- **radius** (float) – An base value for the generation of the 5 pikes star.
- **center** (Pixel) – The wants center from the 5 pikes star.
- **orientation** (float) – An offset in degrees to add to influence the incline of the star.

Return type *Pentagram **

Returns

A pentagram or named pentacle.

note This function generate an 5 extremity star with an centered pentagon from which every vertex go to the center.

see *A blue pentagram*.

Note The radius value is the radius of the base polygon.

hexagram

Hexagram ***generate_hexagram** (float *radius*, Pixel *center*, float *orientation*)

Parameters

- **radius** (float) – An base value for the generation of the 6 pikes star.
- **center** (Pixel) – The wants center from the 6 pikes star.
- **orientation** (float) – An offset in degrees to add to influence the incline of the star.

Return type *Hexagram **

Returns

A hexagram.

note This function generate a 6 extremity star with an centered hexagon from which every vertex go to the center.

see *A blue hexagram*.

Note The radius value is the radius of the base polygon.

Fractal

Polygon ***generate_fractal** (uint32_t *polygon*, float *radius*, Pixel *center*, float *orientation*, bool *open*)

Parameters

- **polygon** (uint32_t) – Base polygon from the fractal.

- **radius** (float) – The radius from the base polygon.
- **center** (Pixel) – The wants center from the fractal.
- **orientation** (float) – An offset in degrees to add to influence the incline of the fractal.
- **open** (bool) – Change the fractal pikes.

Return type *Polygon **

Returns A strange fractal form coming from an crazy brain. :)

Note The radius goes from the center to the farest point of the fratcal (so equal to the `real_length` value) .

Spiral

Spiral ***generate_simple_spiral** (Pixel *center*, uint32_t *turns*, uint32_t *base*, float *offset_exponent*, float *orientation*, bool *reverse*)

Parameters

- **center** (Pixel) – The center from the spiral.
- **turns** (uint32_t) – The number of revolution of the spiral.
- **base** (uint32_t) – the base number of points to make one turn (roundness).
- **offset_exponent** (float) – The factor to compute the distance between 2 points a turn offset.
- **orientation** (float) – An offset in degrees to add to influence the incline of the spiral.
- **reverse** (bool) – Reverse the spiral.

Return type *Spiral **

Returns

A spiral according to the given settings.

see *A blue rounded 3 turns spiral*.

Note The radius goes from the center to the end of the first entire revolution of the spiral.

Note: The `base` and the `offset_exponent` parameters values will influente of the size of the spiral.

You cannot set an radius but the generating function does it.

note The value from the `base` parameter will be divided per two in the resulting spiral.

Warning: The `turns` parameter value will be multiply per 2 to obtains the number of revolutions of the spiral.

The `base` parameter will be divided per two in the resulting spiral.

I can only generate even spirals.

Wheels

Polygon ***generate_wheel** (uint32_t *polygon*, float *radius*, Pixel *center*, float *offset*, float *orientation*)

Parameters

- **polygon** (uint32_t) – Number of sides of the wheel (base polygon).
- **radius** (float) – The radius of the wheel.
- **center** (Pixel) – The center from the wheel.
- **offset** (float) – Size of the peaks of the wheel.

Return type *Polygon **

Returns

A pointed wheel according to the given settings.

note The peaks of the wheel are trigons like a star.

see *A blue 24 peaks wheel*.

Note The radius value is the radius of the base polygon.

Note: You must set an offset value other than 0 because it represent the size of the peaks of the wheel.

The difference betwenn this wheel and a normal star is that is regular.

note the radius Polygon member from this wheel is the radius of the base polygon from this wheel (interior).

Warning: The parameter polygon have a value conform to:

```
360 % polygon == 0
```

Polygon ***generate_circular_saw_wheel** (uint32_t *polygon*, float *radius*, Pixel *center*, float *offset*, float *orientation*, bool *reverse*)

Parameters

- **polygon** (uint32_t) – Number of sides of the circular saw (base polygon).
- **radius** (float) – The radius from the points of the circular saw like wheel.
- **center** (Pixel) – The center from the circular saw like wheel.
- **offset** (float) – Size of the points.
- **reverse** (bool) – Reverse the shift from the circular saw like wheel.

Return type *Polygon **

Returns

A circular saw like pointed wheel.

note This function generate an circular saw like wheel. This is only an polygon with an rectangle triangle on the top of the edges.

see *A blue 24 peaks circular saw*.

Note The radius goes from the center to the end of the pikes (it's equal to the `real_length` value).

> **Warning:** The parameter polygon have a value conform to:
>
> ```
> 360 % polygon == 0
> ```

Polygon ***`generate_wheel_peaks_trigon`**(uint32_t *sides*, float *radius*, Pixel *center*, float *peak_offset*, float *orientation*)

Parameters

- **`sides`** (`uint32_t`) – Number of sides of the base polygon.
- **`radius`** (`float`) – An base value for generating the wheel (rounded polygon).
- **`center`** (`Pixel`) – Center from the wheel (rounded polygon).
- **`peak_offset`** (`float`) – Peak offset.
- **`orientation`** (`float`) – An offset in degrees to add to influence the incline of the wheel (rounded polygon).

Return type *Polygon **

Returns

A pointed wheel (rounded polygon) with peaks implemented as trigon which ends are mini arcs.

note This function generate an wheel (rounded polygon) with peaking as trigons.

see *A blue 24 trigon peaks wheel*.

Note The radius value is the radius of the base polygon.

> **Warning:** The parameter polygon have a value conform to:
>
> ```
> 360 % polygon == 0
> ```

Polygon ***`generate_wheel_peaks_rounded_square`**(uint32_t *sides*, float *radius*, Pixel *center*, float *peak_length*, float *orientation*)

Parameters

- **`sides`** (`uint32_t`) – Number of sides from the wheel (rounded polygon).
- **`radius`** (`float`) – Radius from the wheel (rounded polygon).
- **`center`** (`Pixel`) – Center from the wheel (rounded polygon).
- **`peak_length`** (`float`) – Size of the peaks.
- **`orientation`** (`float`) – An offset in degrees to add to influence the incline of the wheel (rounded polygon).

Return type *Polygon **

Returns

A pointed wheel with peaks looking like a tube.

note This function generate an pointed wheel (rounded polygon) with peaks looking like a tube but they are only right-angled line to the sides and connected trough an arc.

see *A blue 24 rounded square peaks wheel*.

Note The radius value is the radius of the base polygon.

Warning: The parameter sides have a value conform to:

```
360 % sides == 0 and sides <= 24
```

CHAPTER 5

Displaying forms

display functions

Here are describe the Form display utlities.

Line

int **display_line**(SDL_Renderer **pRenderer*, Line **line*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **line** (Line *) – The line to display.

Return type int

Returns

0 on success, -1 on failure.

see *display line*.

This function display the line in the current color and at the current position.

Arc

int **display_arc**(SDL_Renderer **pRenderer*, Arc **arc*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **arc** (Arc *) – The arc to display.

Return type int

Returns

0 on success, -1 on failure.

see *display arc*.

This function display the arc in the current color and at the current position.

Polygon

int **display_polygon**(SDL_Renderer **pRenderer*, Form **polygon*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **polygon** (`Form *`) – The polygon to display.

Return type `int`

Returns

0 on success, -1 on failure.

see *display polygon*.

This function display the polygon lined in the current color and at the current position.

int **display_strikethrough_polygon**(SDL_Renderer **pRenderer*, Form **polygon*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **polygon** (`Form *`) – The polygon to display.

Return type `int`

Returns

0 on success, -1 on failure.

see *display strikethrough polygon*.

This function display the polygon lined strikethrough in the current color and at the current position.

display_filled_polygon(SDL_Renderer **pRenderer*, Form **polygon*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **polygon** (`Form *`) – The polygon to display.

Return type `int`

Returns

0 on success, -1 on failure.

see *display filled polygon*.

This function display the polygon filled in the current color and at the current position.

Pentagram & Hexagram

int **display_pentagram**(SDL_Renderer **pRenderer*, Pentagram **pentagram*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **pentagram** (`Pentagram *`) – The pentagram to display.

Return type `int`

Returns

0 on success, -1 on failure.

see *display pentagram*.

This function display the pentagram in the current color and at the current position.

int **display_hexagram**(SDL_Renderer **pRenderer*, Pentagram **hexagram*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **hexagram** (`Hexagram *`) – The hexagram to display.

Return type `int`

Returns

0 on success, -1 on failure.

see *display hexagram*.

This function display the hexagram in the current color and at the current position.

Star

int **display_star**(SDL_Renderer **pRenderer*, Star **star*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **star** (`Star *`) – The star to display.

Return type `int`

Returns

0 on success, -1 on failure.

see *display star*.

This function display the star in the current color and at the current position.

int **display_flower_star**(SDL_Renderer **pRenderer*, Star **star*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **star** (`Star *`) – The star to display.

Return type `int`

Returns

0 on success, -1 on failure.

see *display flower star*.

This function display the flower star in the current color and at the current position.

int **display_strikethrough_star**(SDL_Renderer **pRenderer*, Star **star*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **star** (Star *) – The star to display.

Return type int

Returns

0 on success, -1 on failure.

see *display strikethrough star*.

This function display the strikethrough star in the current color and at the current position.

int **display_polygon_star**(SDL_Renderer **pRenderer*, Star **star*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **star** (Star *) – The star to display.

Return type int

Returns

0 on success, -1 on failure.

see *display polygon star*.

This function display the polygon star in the current color and at the current position.

int **display_filled_star**(SDL_Renderer **pRenderer*, Star **star*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **star** (Star *) – The star to display.

Return type int

Returns

0 on success, -1 on failure.

see *display filled star*.

This function display the strikethrough star in the current color and at the current position.

Spiral

int **display_spiral**(SDL_Renderer **pRenderer*, Spiral **spiral*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.

- **`spiral`** (`Spiral *`) – The spiral to display.

Return type `int`

Returns

0 on success, -1 on failure.

see *display spiral*.

This function display the spiral in the current color and at the current position.

Anti-aliasing display functions

Line

int **`aa_display_line`**(SDL_Renderer **pRenderer*, Line **line*)

Parameters

- **`pRenderer`** – A SDL_Renderer pointer.
- **`line`** (`Line *`) – The line to display.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the line in the current color and at the current position.

Arc

int **`aa_display_arc`**(SDL_Renderer **pRenderer*, Arc **arc*)

Parameters

- **`pRenderer`** – A SDL_Renderer pointer.
- **`arc`** (`Arc *`) – The arc to display.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the arc in the current color and at the current position.

Polygon

int **`aa_display_polygon`**(SDL_Renderer **pRenderer*, Form **polygon*)

Parameters

- **`pRenderer`** – A SDL_Renderer pointer.
- **`polygon`** (`Form *`) – The polygon to display.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the polygon lined in the current color and at the current position.

int **aa_display_strikethrough_polygon**(SDL_Renderer **pRenderer*, Form **polygon*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **polygon** (`Form *`) – The polygon to display.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the polygon lined strikethrough in the current color and at the current position.

Pentagram & Hexagram

int **aa_display_pentagram**(SDL_Renderer **pRenderer*, Pentagram **pentagram*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **pentagram** (`Pentagram *`) – The pentagram to display.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the pentagram in the current color and at the current position.

int **aa_display_hexagram**(SDL_Renderer **pRenderer*, Pentagram **hexagram*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **hexagram** (`Hexagram *`) – The hexagram to display.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the hexagram in the current color and at the current position.

Star

int **aa_display_star**(SDL_Renderer **pRenderer*, Star **star*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **star** (`Star *`) – The star to display.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the star in the current color and at the current position.

int **aa_display_flower_star**(SDL_Renderer **pRenderer*, Star **star*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **star** (`Star *`) – The star to display.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the flower star in the current color and at the current position.

int **aa_display_strikethrough_star**(SDL_Renderer *pRenderer, Star *star)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **star** (`Star *`) – The star to display.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the strikethrough star in the current color and at the current position.

int **aa_display_polygon_star**(SDL_Renderer *pRenderer, Star *star)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **star** (`Star *`) – The star to display.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the polygon star in the current color and at the current position.

Spiral

int **aa_display_spiral**(SDL_Renderer *pRenderer, Spiral *spiral)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **spiral** (`Spiral *`) – The spiral to display.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the spiral in the current color and at the current position.

Thickness settable display functions

Line

int **display_line_thick**(SDL_Renderer *pRenderer, Line *line, uint8_t thickness)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **line** (`Line *`) – The line to display.
- **thickness** (`uint8_t`) – The line width.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the line in the current color and at the current position.

Arc

int **`display_arc_thick`**(SDL_Renderer **pRenderer*, Arc **arc*, uint8_t *thickness*)

Parameters

- **`pRenderer`** – A SDL_Renderer pointer.
- **`arc`** (`Arc *`) – The arc to display.
- **`thickness`** (`uint8_t`) – The line width.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the arc in the current color and at the current position.

Polygon

int **`display_polygon_thick`**(SDL_Renderer **pRenderer*, Form **polygon*, uint8_t *thickness*)

Parameters

- **`pRenderer`** – A SDL_Renderer pointer.
- **`polygon`** (`Form *`) – The polygon to display.
- **`thickness`** (`uint8_t`) – The line width.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the polygon lined in the current color and at the current position.

int **`display_strikethrough_polygon_thick`**(SDL_Renderer **pRenderer*, Form **polygon*, uint8_t *thickness*)

Parameters

- **`pRenderer`** – A SDL_Renderer pointer.
- **`polygon`** (`Form *`) – The polygon to display.
- **`thickness`** (`uint8_t`) – The line width.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the polygon lined strikethrough in the current color and at the current position.

Pentagram & Hexagram

int **`display_pentagram_thick`**(SDL_Renderer **pRenderer*, Pentagram **pentagram*, uint8_t *thickness*)

Parameters

- **`pRenderer`** – A SDL_Renderer pointer.

- **pentagram** (`Pentagram *`) – The pentagram to display.
- **thickness** (`uint8_t`) – The line width.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the pentagram in the current color and at the current position.

int **display_hexagram_thick** (SDL_Renderer **pRenderer*, Pentagram **hexagram*, uint8_t *thickness*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **hexagram** (`Hexagram *`) – The hexagram to display.
- **thickness** (`uint8_t`) – The line width.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the hexagram in the current color and at the current position.

Star

int **display_star_thick** (SDL_Renderer **pRenderer*, Star **star*, uint8_t *thickness*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **star** (`Star *`) – The star to display.
- **thickness** (`uint8_t`) – The line width.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the star in the current color and at the current position.

int **display_flower_star_thick** (SDL_Renderer **pRenderer*, Star **star*, uint8_t *thickness*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **star** (`Star *`) – The star to display.
- **thickness** (`uint8_t`) – The line width.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the flower star in the current color and at the current position.

int **display_strikethrough_star_thick** (SDL_Renderer **pRenderer*, Star **star*, uint8_t *thickness*)

Parameters

- **pRenderer** – A SDL_Renderer pointer.
- **star** (`Star *`) – The star to display.
- **thickness** (`uint8_t`) – The line width.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the strikethrough star in the current color and at the current position.

int **`display_polygon_star_thick`** (SDL_Renderer **pRenderer*, Star **star*, uint8_t *thickness*)

Parameters

- **`pRenderer`** – A SDL_Renderer pointer.
- **`star`** (`Star *`) – The star to display.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the polygon star in the current color and at the current position.

Spiral

int **`display_spiral_thick`** (SDL_Renderer **pRenderer*, Spiral **spiral*, uint8_t *thickness*)

Parameters

- **`pRenderer`** – A SDL_Renderer pointer.
- **`spiral`** (`Spiral *`) – The spiral to display.
- **`thickness`** (`uint8_t`) – The line width.

Return type `int`

Returns 0 on success, -1 on failure.

This function display the spiral in the current color and at the current position.

Define your own displaying functions

Note: Define your own displaying functions.

You can implement your own displaying functions for per example:

- Display every line from a polygon in an different color.
- Display severals polygons filled overlaps in differents tones from a color with sinking alpha value.

And whatever your brain can imagined.

CHAPTER 6

Operations

Here are presented the functions which apply an transformation operation on a *Pixel* or on a *Form*.

Pixel operations functions

This function are the base for the forms operations functions.

You can use it to implement your own operations.

Pixel **rotate**(Pixel *center*, float *angle*, Pixel *pixel*)

Parameters

- **center** (Pixel) – The center of the rotation.
- **angle** (float) – The angle of the rotation in degress.
- **pixel** (Pixel) – The pixel to rotate.

Return type *Pixel*.

Returns A new rotated *Pixel*.

This function return the rotated pixel from *angle degrees* around the given center in clock sens.

Note: Rotating a **pixel** around the **origin** is easy doing according following formel in **matrix** form:

```
+-  -+      +--                             --+      +- -+
| x1 |      | cos(angle)   -sin(angle)    0   |      | x |
| y2 |  =   | sin(angle)    cos(angle)    0   |  *   | y |
| 0  |      |     0             0         1   |      | 0 |
+-  -+      +--                             --+      +- -+
```

So you can simply **translate** *the pixel in accord to the origin* **rotate** it and **translate** it **back**.

But **SDL2_gfxutils** use a function based on matrix to **rotate** the pixel **around** an **arbitrary point**.

See the source at file `base_functions.c`.

Pixel **`scale`** (Pixel *center*, float *factor*, Pixel *pixel*)

Parameters

- **`center`** (`Pixel`) – The center of the form to scale.
- **`factor`** (`float`) – The scaling factor.
- **`pixel`** (`Pixel`) – The pixel to scale.

Return type *Pixel*.

Returns A new scaled pixel (position).

This function return the new position from pixel scaled by factor:

```
factor < 1 == scaling littler.

factor > 1 == scaling greater.
```

Note: A pixel can only be **corrected scaled** in accord **to** the **origin**.

So you can simply **translate** *the pixel in accord to the origin* multiply `x` and `y` with the **scaling** factor and **translate** it **back**.

Pixel **`translate`** (Pixel *pixel*, float *x*, float *y*)

Parameters

- **`pixel`** (`Pixel`) – The pixel to translate.
- **`x`** (`float`) – The translation value from the x axes (even negativ).
- **`y`** (`float`) – The translation value from the y axes (even negativ).

Return type *Pixel*.

Returns A new pixel translated from `x` and `y`.

This function translate a pixel from value `x` and `y`.

Note `x` and **y** can be negativ for translating in direction of the left or to the top.

Note: For translating a pixel simply **add** (*even negativ*) the wanted values to the `x` and `y` from the pixel *Pixel* members.

Pixel **`mirror`** (Pixel *pixel*, Pixel *center*, char *axes*)

Parameters

- **`pixel`** (`Pixel`) – The pixel to mirror.
- **`center`** (`Pixel`) – The center of the mirroring.
- **`axes`** (`char`) – 'X' or 'Y'.

Return type *Pixel*.

Returns A new pixel mirrored around center trough the X or Y axes.

This function mirror a pixel through the `x` (**Vertical**) or `y` (**Horizontal**) axes in relationship to the given center.

Note: The center of the mirroring.

The center argument given the mirroring center and in case of mirroring on the:

- X axes only the `x` of the *Pixel* counting.
- Y axes only the `y` of the *Pixel* counting.

Warning: Take care of the *Form* operation function condition.

Operations on a pixel according to the origin

```
x += -center.x ; y += -center.y  // (translate according the origin).

// Operation on the pixel.

x += center.x ; y += center.y  // (translate it back).
```

Forms operations functions

They all apply a transformation on a form by calling the pixels operations functions.

By using a pointer on the generic *Form* type form to transform given as argument.

void **`rotate_form`**(Form **form*, float *angle*)

Parameters

- **`form`** (`Form`) – A pointer on the form to rotate.
- **`angle`** (`float`) – The angle of the rotation.

Return type `void`.

This function perform a **rotation** on a `form` itself, through an **pointer** on it, from *angle degrees* around the `center` from the `form`.

Note: Rotation center.

You can change temporary the center of the form you want to rotate the form around the wanted center,

with the function *set_form_center()*.

instead of the center of the form itself.

warning If you use a display function which strikethrough from the center: the displaying will degenerate (maybe you do it express).

void **`scale_form`**(Form **form*, float *factor*)

Parameters

- **`form`** (`Form`) – A pointer on the form to scale.
- **`factor`** (`float`) – The scaling factor.

Return type `void`.

This function **scale** the adressed `form` from value `factor`.

Note: Scaling factor.

•if factor > 1.0 the size of the form increase.

•if factor < 1.0 the size from the form decrease.

Note: You can set a new radius (which will update the `length` *Form* type member) directly,

With the function *set_form_radius()*

What permit to change the size of the form without using a factor but a radius instead.

warning **Use only integers values** or not more than **3 precision** (`%.3f`) otherwise your request will not be exactly satisfy.

---

void **`translate_form`**(Form **form*, float *x*, float *y*)

Parameters

- **`form`** (`Form`) – A pointer on the form to translate.
- **`x`** (`float`) – The translation value from the x axes (even negativ).
- **`y`** (`float`) – The translation value from the y axes (even negativ).

Return type `void`.

This function **translate** the adressed `form` from values `x` and `y`.

Note **x** and **y** can be negativ for translating in direction of the left or to the top.

Warning Use only integers values or not more than 3 precision (%.3f) otherwise your request will not be exactly satisfy.

void **`mirror_form`**(Form **form*, Pixel *center*, char *axes*)

Parameters

- **`form`** (`Pixel`) – A pointer on the form to mirror.
- **`center`** – The center for the mirroring.
- **`axes`** (`char`) – ‘X’ or ‘Y’.

Return type `void`.

This function **mirror** the given `form` through the `x` (**Vertical**) or `y` (**Horizontal**) axes in relationship to the given center.

Warning: This function is subject of a big condition to work properly !!!

All coordinates must be at one side from the center axe.

Argument axes:

'X') If mirroring over axes **X** all pixels must must be **above** or **below** from the `center` argument `x` *Pixel* type member.

'Y') If mirroring over axes **Y** all pixels must must be at the **right** or at the **left** from the `center` argument `y` *Pixel* type member.

`Form *remove_doubles_form(Form *form) ;`

Parameters

- **`form`** (`Form`) – A pointer on the form to mirror.

Return type *Form **

Returns

The same form with doubles (same values) coordinates removed.

note The given form is free and reallocated (sorry can't do otherwise).

CHAPTER 7

Setters functions

This functions will permit you to change and so transform and or animating your forms.

Or to change their colors.

Center

void **set_form_center** (Form **form*, Pixel *center*, bool *translate*)

Parameters

- **form** (Form) – The form to set a new center.
- **center** (Pixel) – The new center to set.
- **translate** (bool) – Translating all coordinates according the new center.

Return type void

Returns void

This function set a **new** center to the given form with or without **translating** all coordinates from the *Form* according the **new** center.

Radius (size)

void **set_form_radius** (Form **form*, float *radius*)

Parameters

- **form** (Form) – The form to set a new radius.
- **radius** (float) – The new radius to set.

Return type void

Returns `void`

This function set a **new** `radius` to the given `form`.

What permit to change the *Form* **size** directly by setting a new `radius` by given a value and not a **scaling** `factor` as in the *scale_form()* function.

Warning **Use only integers values** or not more than **3 precision** (`%.3f`) in the **radius**, *Form* generating functions, argument.

Color

void **set_form_color**(Form **form*, uint8_t *red*, uint8_t *green*, uint8_t *blue*, uint8_t *alpha*)

Parameters

- **form** (`Form`) – The *Form* to change the color from.
- **red** (`uint8_t`) – a value between 0 and 255.
- **green** (`uint8_t`) – a value between 0 and 255.
- **blue** (`uint8_t`) – a value between 0 and 255.
- **alpha** (`uint8_t`) – a value between 0 and 255.

Return type `void`

Returns `void`

This function set a **new** color by updating the *Color* type members from the given `form`.

void **set_line_color**(Line **line*, uint8_t *red*, uint8_t *green*, uint8_t *blue*, uint8_t *alpha*)

Parameters

- **form** (`Form`) – The *Line* to change the color from.
- **red** (`uint8_t`) – a value between 0 and 255.
- **green** (`uint8_t`) – a value between 0 and 255.
- **blue** (`uint8_t`) – a value between 0 and 255.
- **alpha** (`uint8_t`) – a value between 0 and 255.

Return type `void`

Returns `void`

This function set a **new** color by updating the *Color* type members from the given `line`.

CHAPTER 8

Getters functions

This functions are **convienence** functions to get the **current value** of the **generic** *Form* type **members**.

note But you can easily access the members directly with the -> pointer notation.

Center

Pixel **`get_form_center`**(Form **form*)

Parameters

- **`form`** (`Form`) – The `form` to get the center from.

Return type `Pixel`

Returns The current center *Pixel* member of the given `form` from type *Form*.

Color

Color **`get_form_color`**(Form **form*)

Parameters

- **`form`** (`Form`) – The `form` to get the `Color` from.

Return type `Color`

Returns The current *Color* member of the given `form` from type *Form*.

Length

float **get_form_length** (Form **form*)

Parameters

- **form** (Form) – The form to get the length from.

Return type float

Returns The current length member of the given form from type *Form*.

Note: The member named length is very often the **radius** from the form.

See the *Forms generating functions* page for the what the length member represent.

Or in other words how it is compute from the often, radius argument value.

Real length

float **get_form_real_length** (Form **form*)

Parameters

- **form** (Form) – The form to get the real length from.

Return type float

Returns The current real_length member of the given form from type *Form*.

Note: The member named real_length is the distance between the **center** and the **farest** *coordinates* from the **center**.

note It can be used per example to build a bounding box from a polygon for collision detection or the purpose you want.

Warning The member name real_length is not always exactly after executing *set_form_radius()* but nearly approximate.

Orientation

float **get_form_orientation** (Form **form*)

Parameters

- **form** (Form) – The form to get the length from.

Return type float

Returns The current orientation member of the given form from type *Form*.

Note: The member and argument named orientation is always given as argument.

It represent the incline of the forms according *the angle measurement convention*.

You can use the `orientation` argument value to rotate a form if you generate and destroy the *Form*.

CHAPTER 9

Miscellaneous

Here are presented the trajectories computing function.

trajectories

void **compute_trajectory** (Pixel *positions[]*, Line *trajectory*, uint32_t *steps*)

Parameters

- **positions** – An array from type Pixel from size steps.
- **trajectory** – A line on where settings the pixels along.
- **steps** – The number of pixel to set along the line trajectory.

Return type `void`

This function feed the array `positions[steps]` by computing the number of pixels `steps` along the *Line line* at equal distance.

Warning: You can declare the positions array or allocated it dynamicaly:

```
uint32_t steps = 32 ;

Pixel positions[steps] ;

/** Or **/

Pixel *positions = (Pixel *) calloc((ssize_t) steps, sizeof(Pixel)) ;
```

Note: Guideline animation.

After executing this function you get an array of pixels with which you can implement an animation with the array of pixels as guideline.

CHAPTER 10

Utils

Check

void **check_renderer** (SDL_Renderer **pRenderer*)

Parameters

- **pRenderer** (*SDL_Renderer*) – The SDL2 Renderer

Return type void

Returns void

Check the validity of the SDL_Renderer.

```
if (pRenderer == NULL) {
  fprintf(stderr,"SDL Renderer error (%s)\n",SDL_GetError());
  exit(EXIT_FAILURE) ;
}
```

Note: Function used in all *displaying functions*.

void **check_form** (Form **form*)

Parameters

- **form** (Form) – the form to check.

Return type void

Returns void

Check only if the given parameter is equal to NULL.

```
if (form == NULL) {
  fprintf(stderr,"Invalid form argument !\n");
  exit(EXIT_FAILURE) ;
}
```

Note: Function used in all *forms setters functions* and all *forms operations functions*.

Memory

Form ***new_form**(uint32_t *count*)

Parameters

- **pRenderer** (`uint32_t`) – the number of coordinates pair to allocate.

Return type *Form*

Returns A new allocated *Form*

This function allocate the required space for the given `count` argument number of coordinates arrays.

And set the `count` member from the returned *Form*.

Warning The other members must you set yourself.

void **free_form**(Form **form*)

Parameters

- **form** (`Form`) – the `form` to free.

Return type `void`

Returns `void`

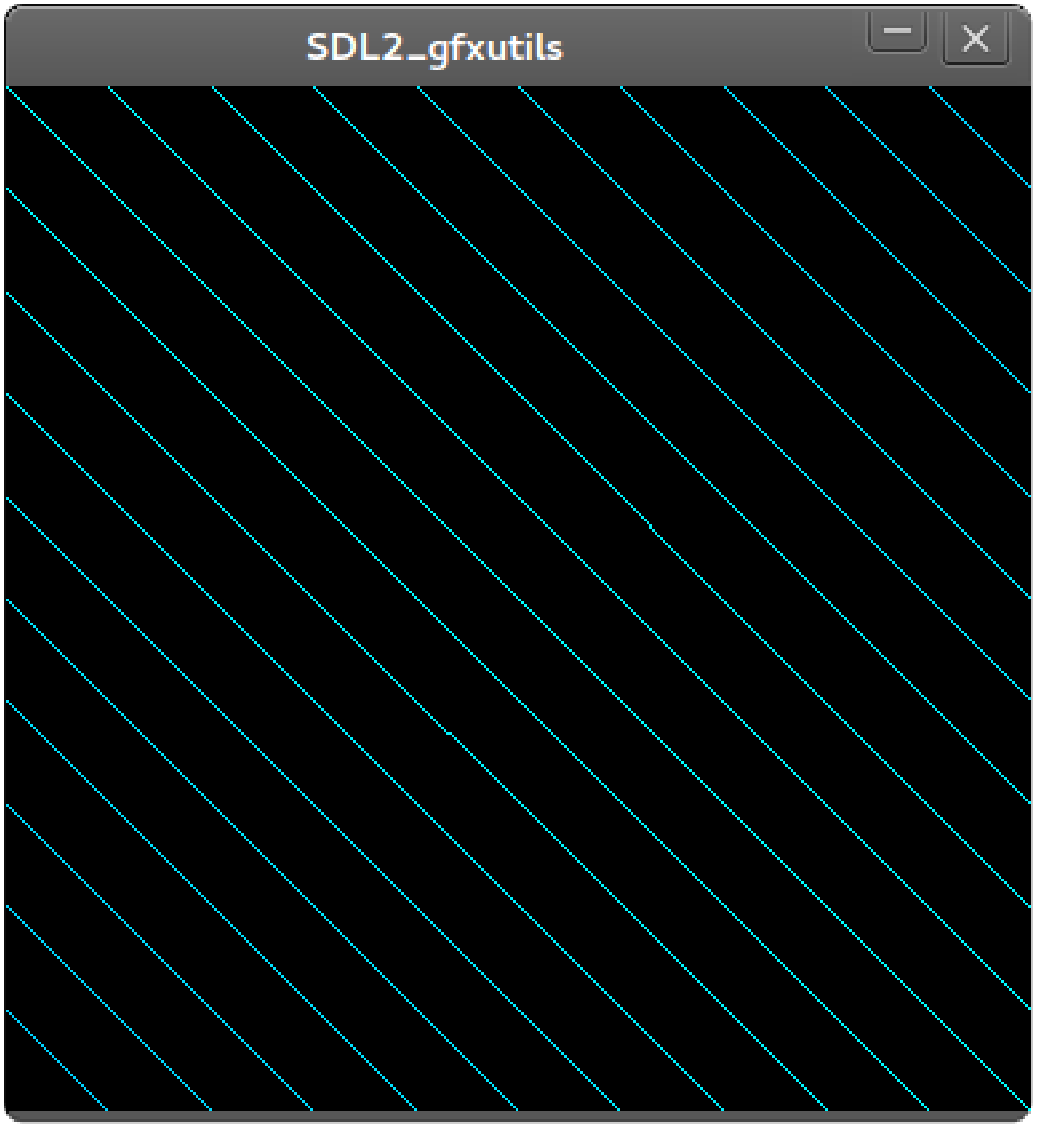
This function free the allocated coordinates arrays from the *Form*, free the `form` pointer and set it on `NULL`.

CHAPTER 11

SDL2_gfxutils Images gallery

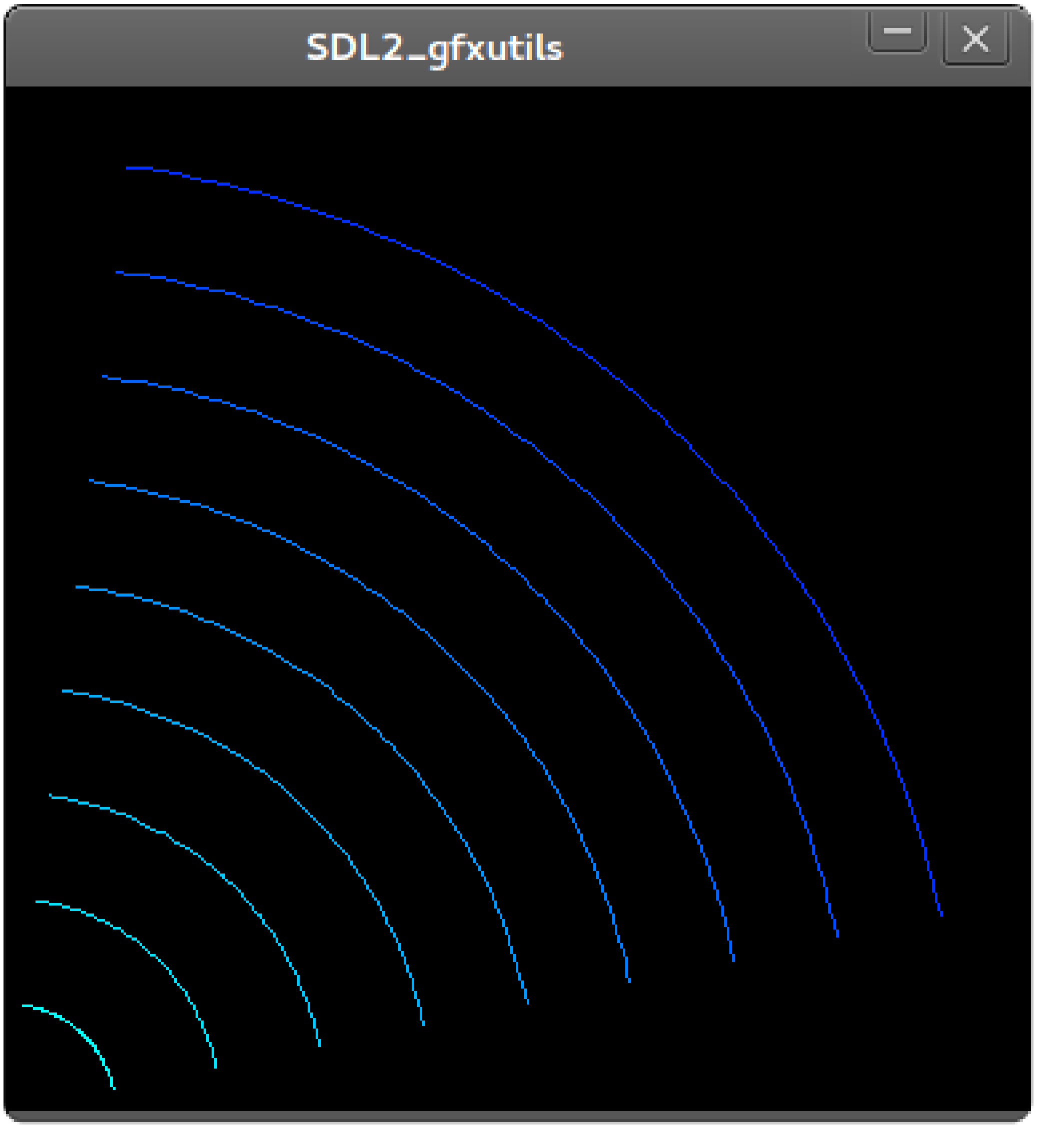
generate segment

- display line



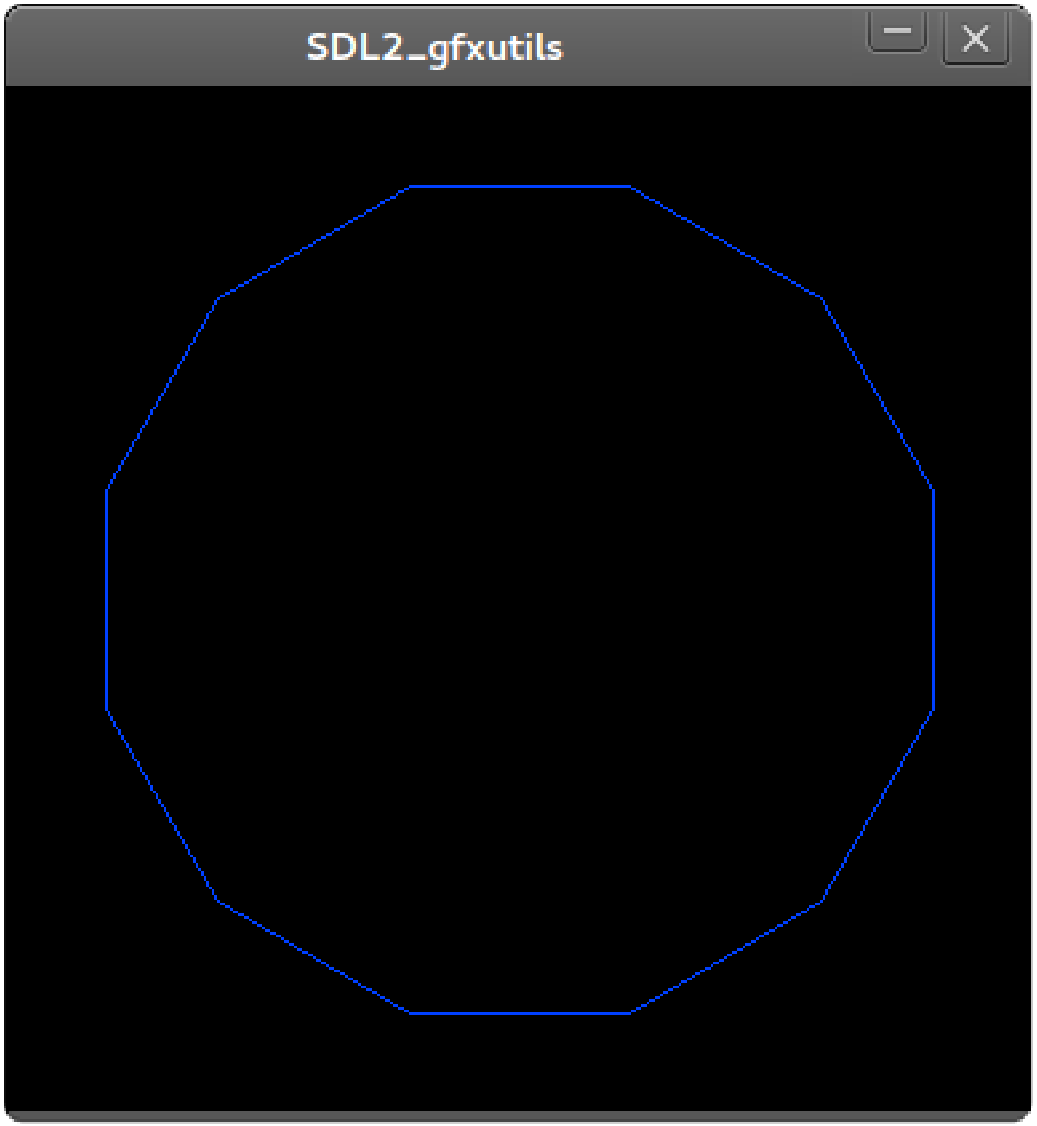
generate circle arc

- display arc

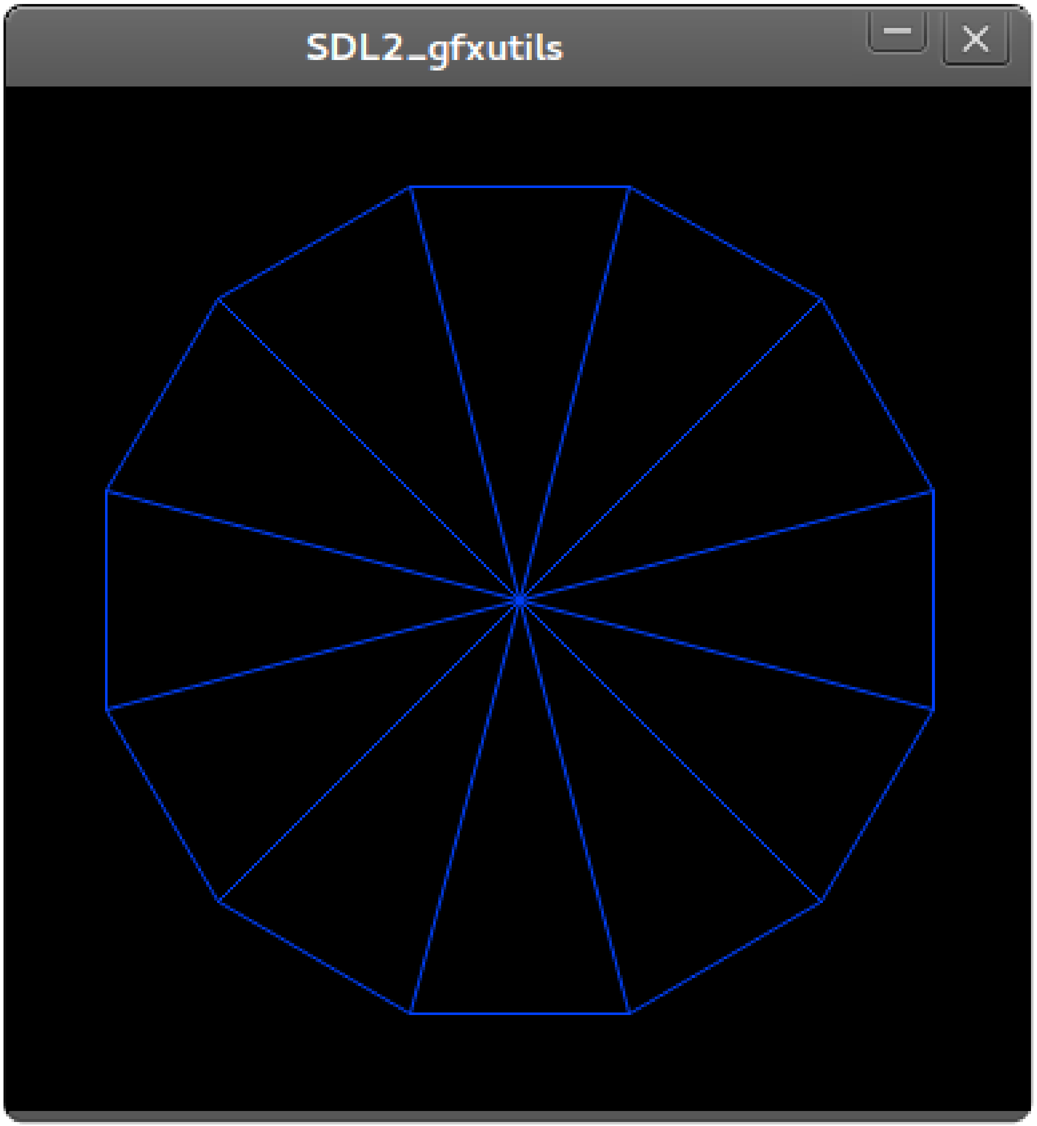


generate polygon radius

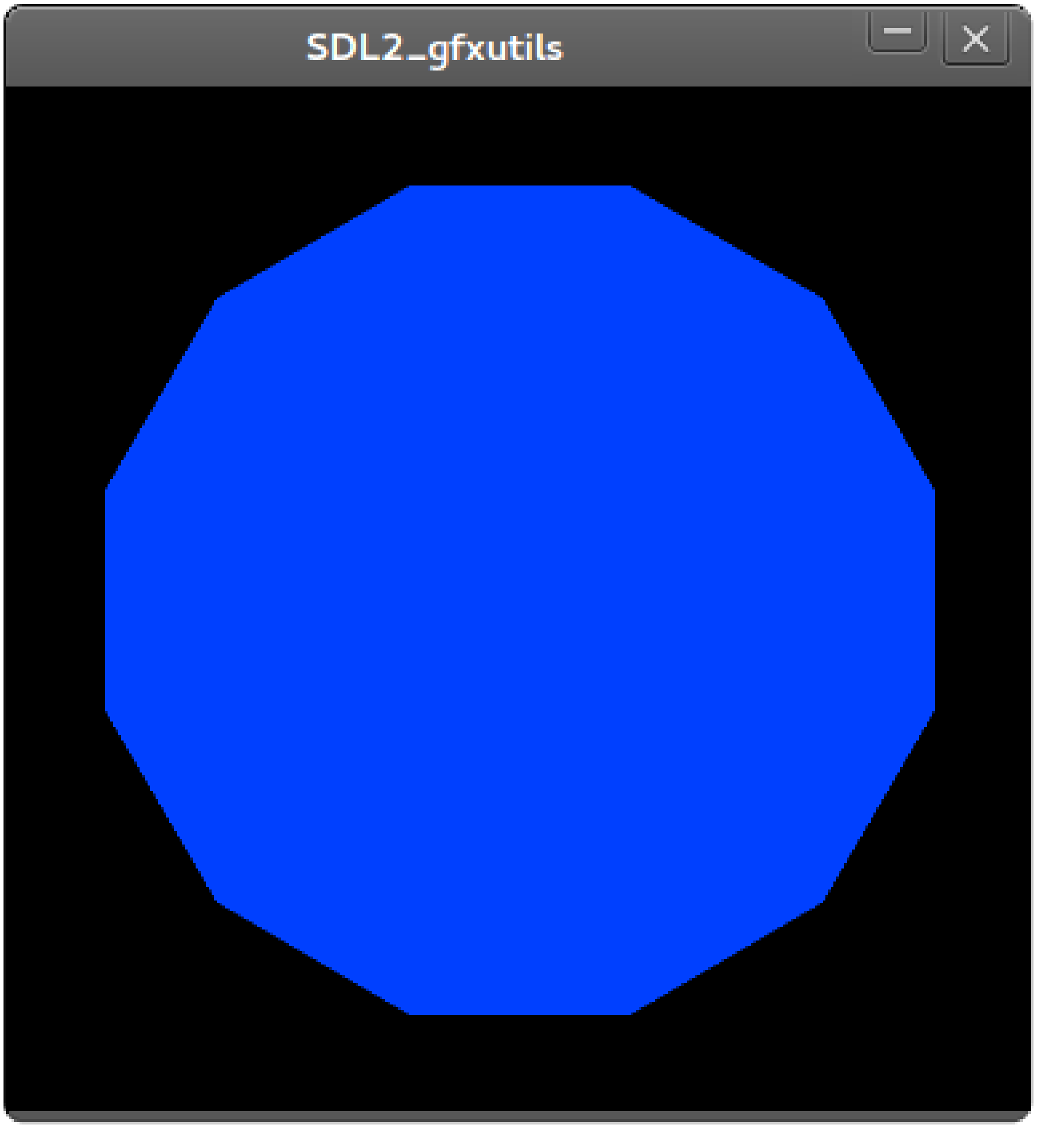
- display polygon



- display strikethrough polygon

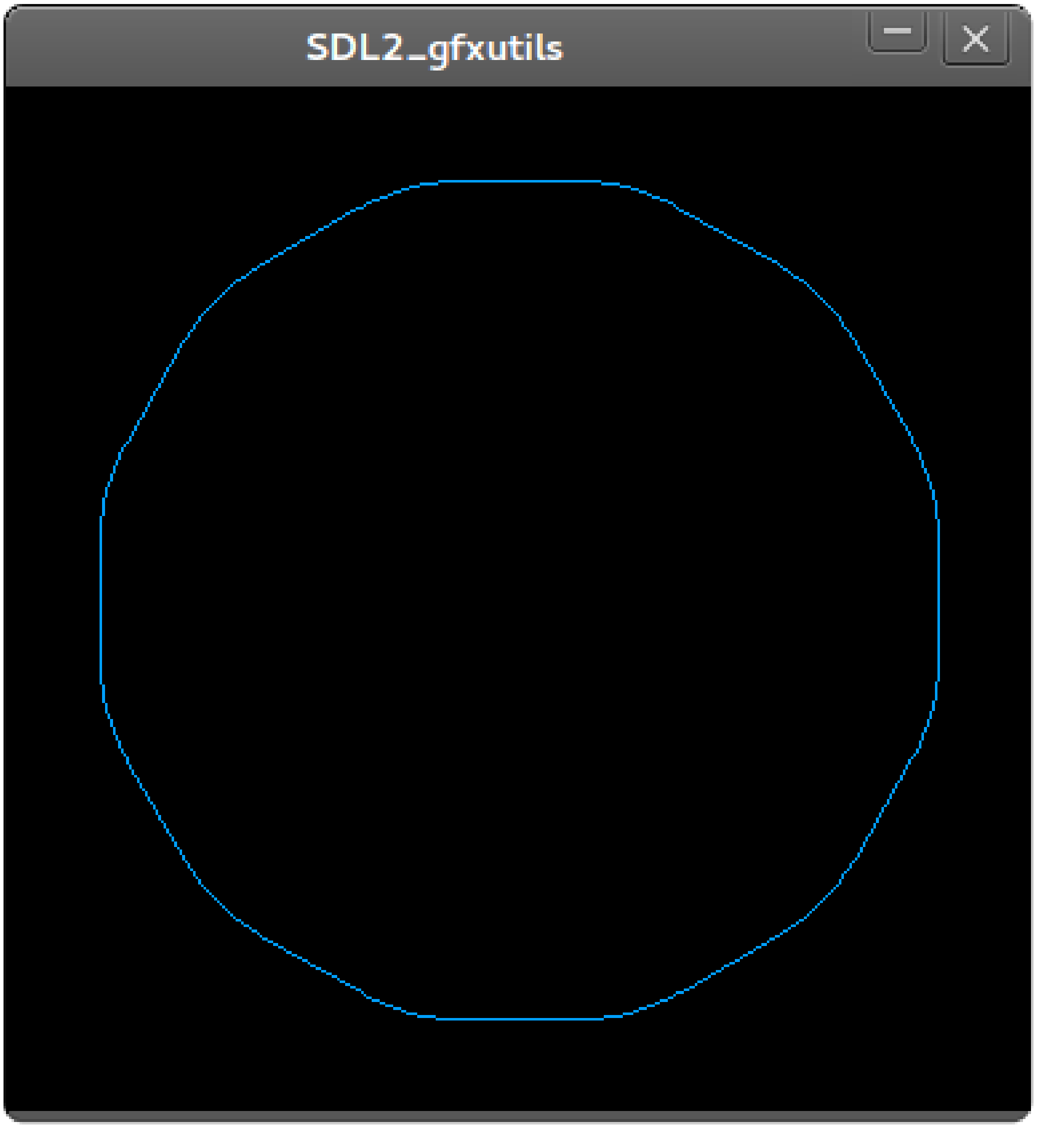


- display filled polygon

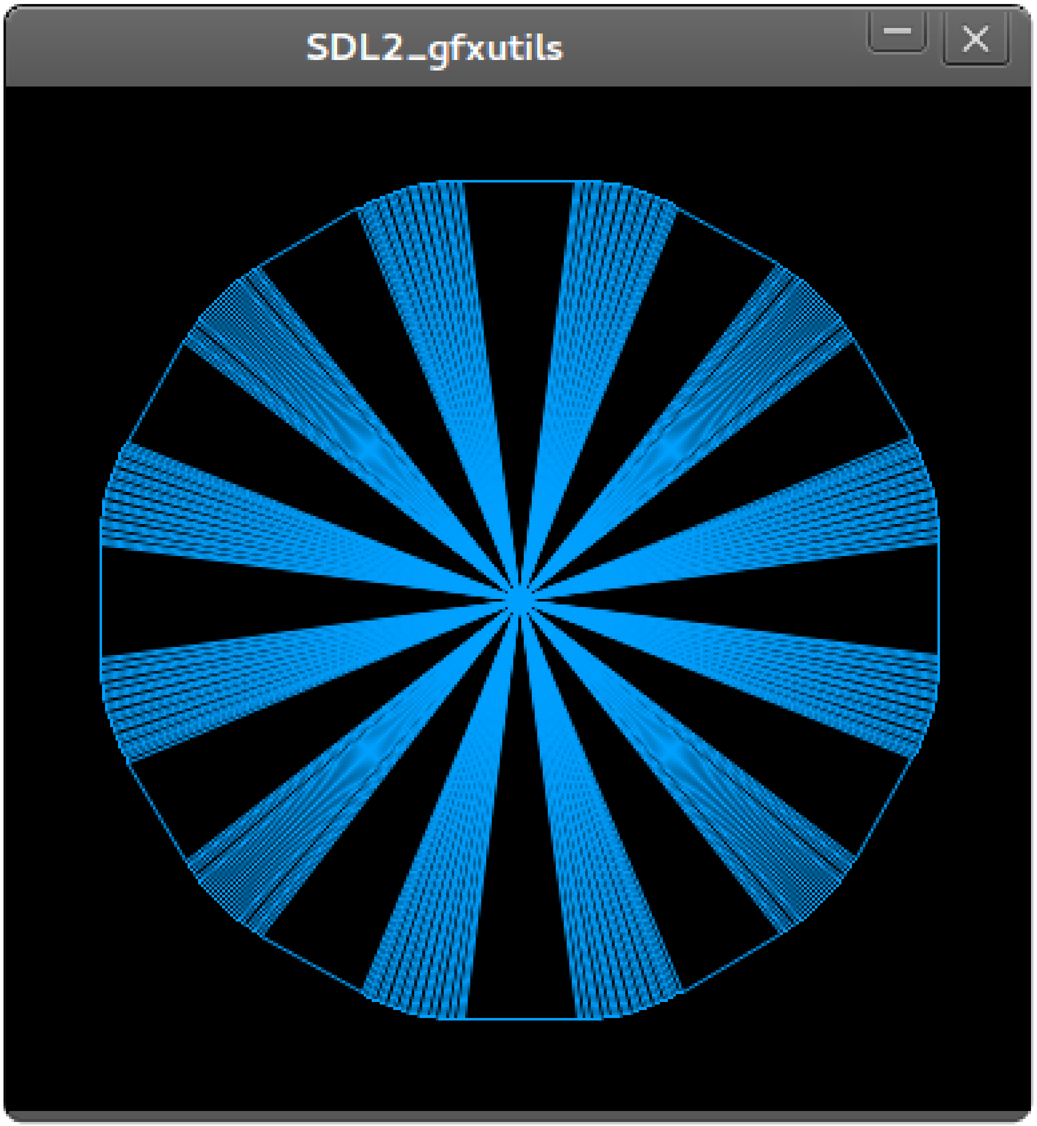


generate corners rounded polygon

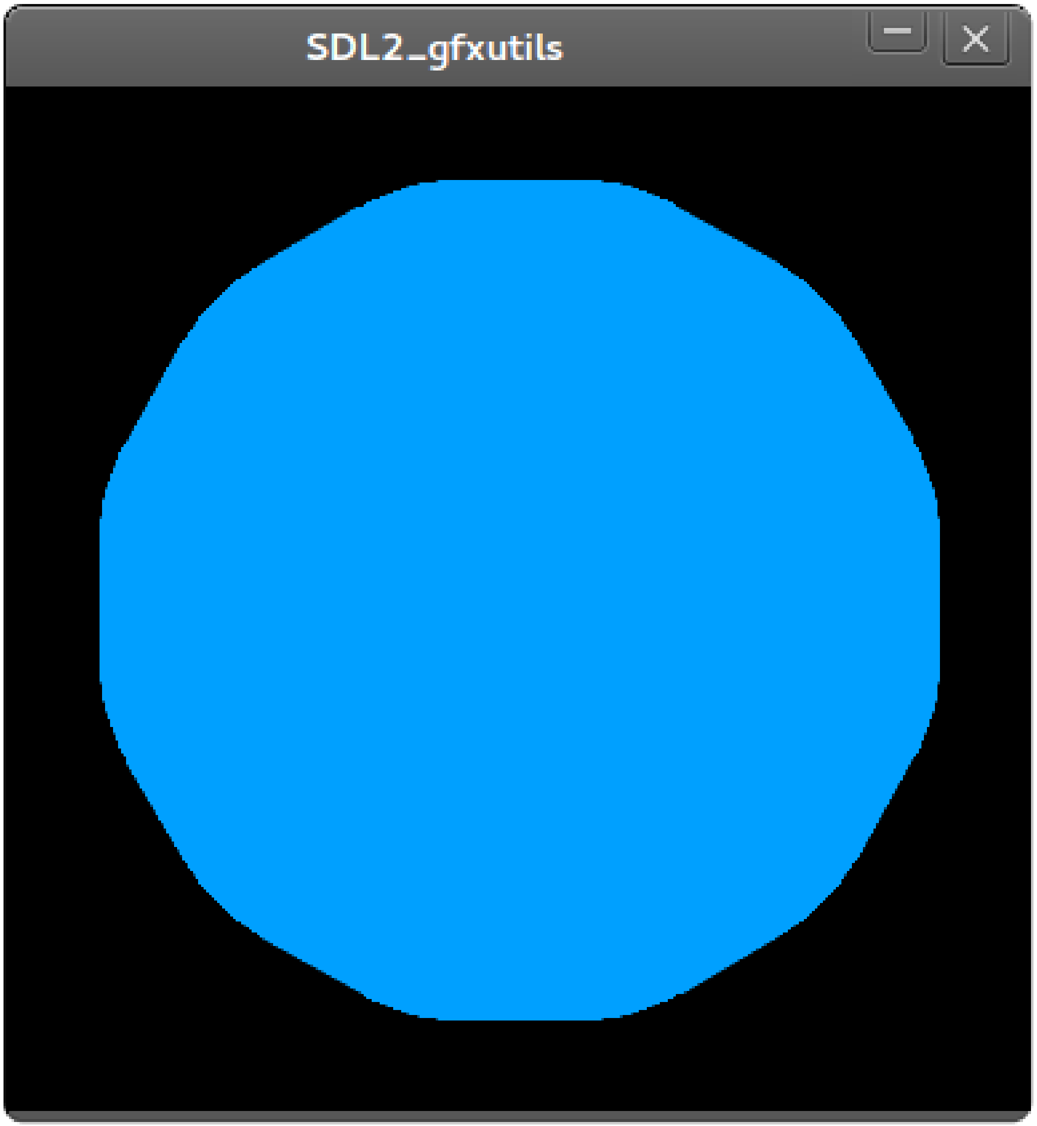
- display polygon



- display strikethrough polygon

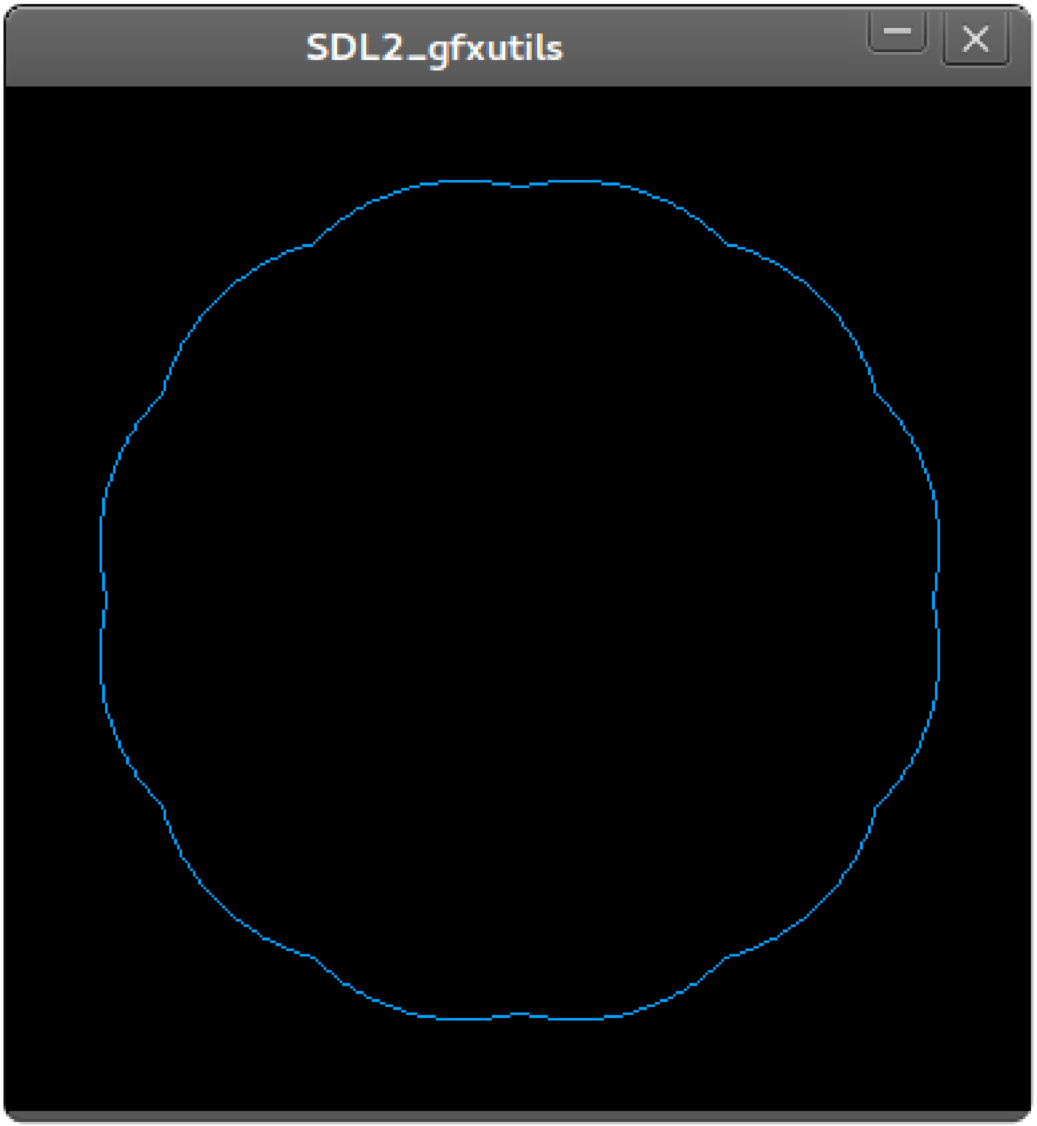


- display filled polygon

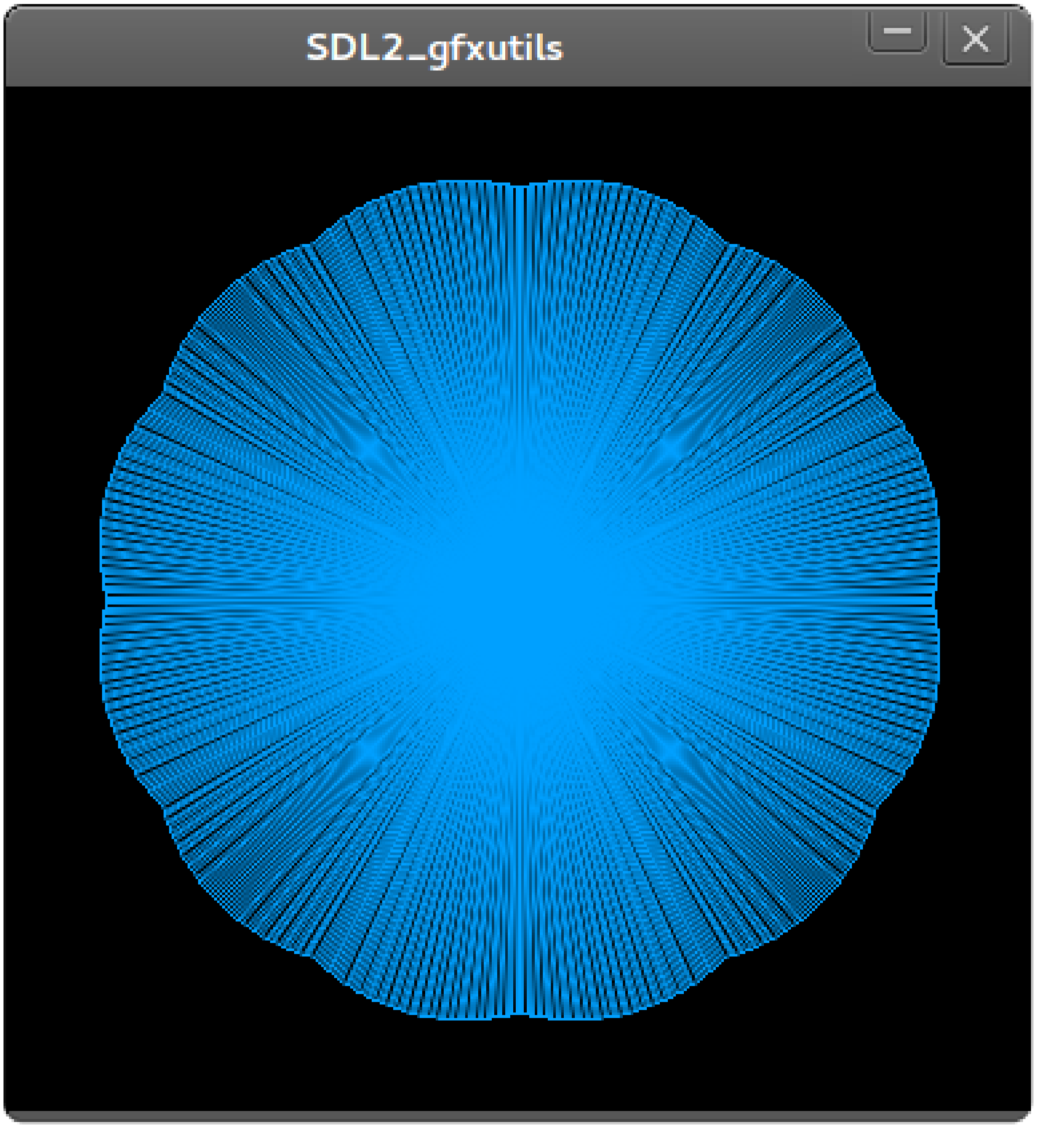


generate sides rounded polygon

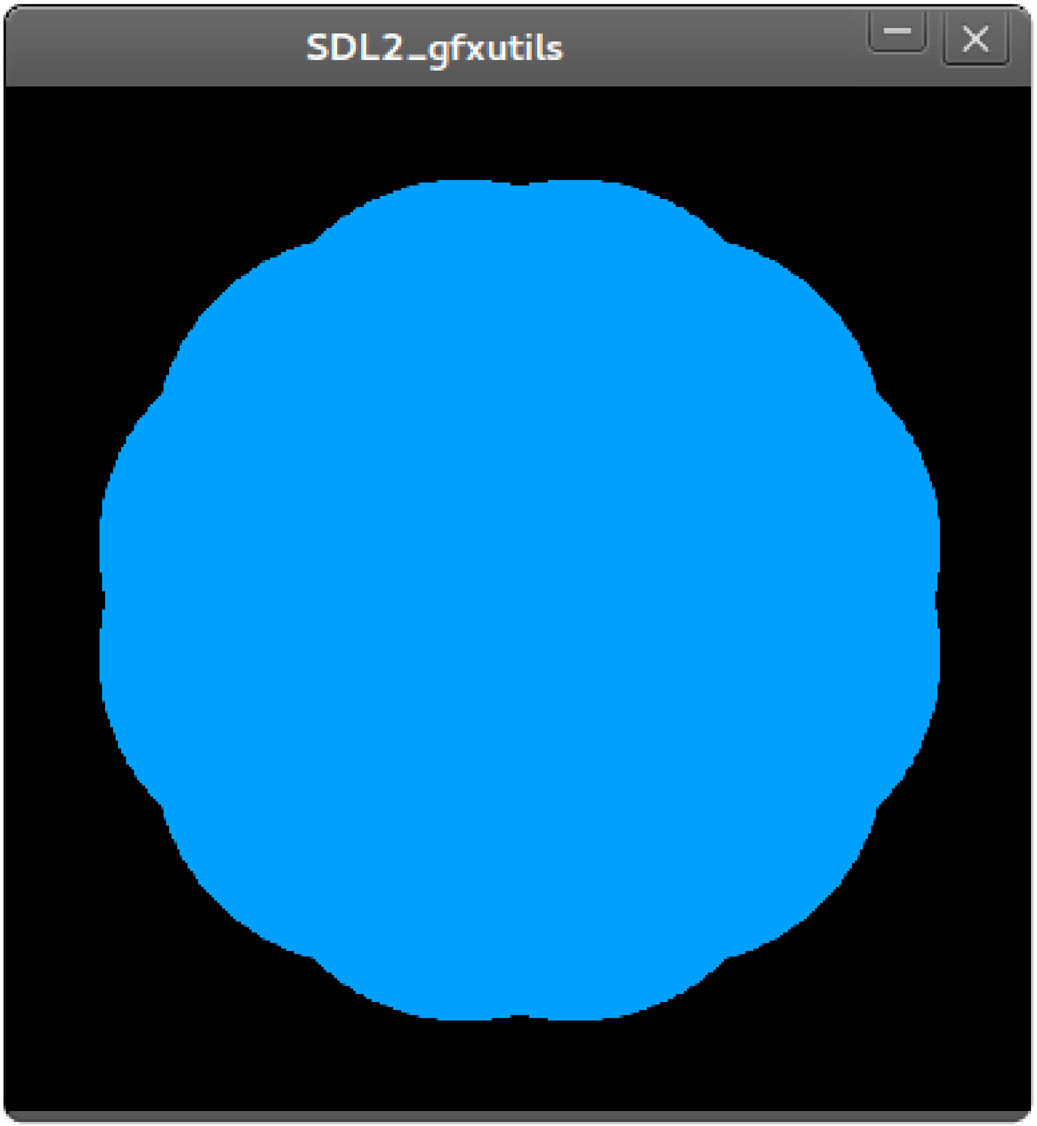
- display polygon



- display strikethrough polygon

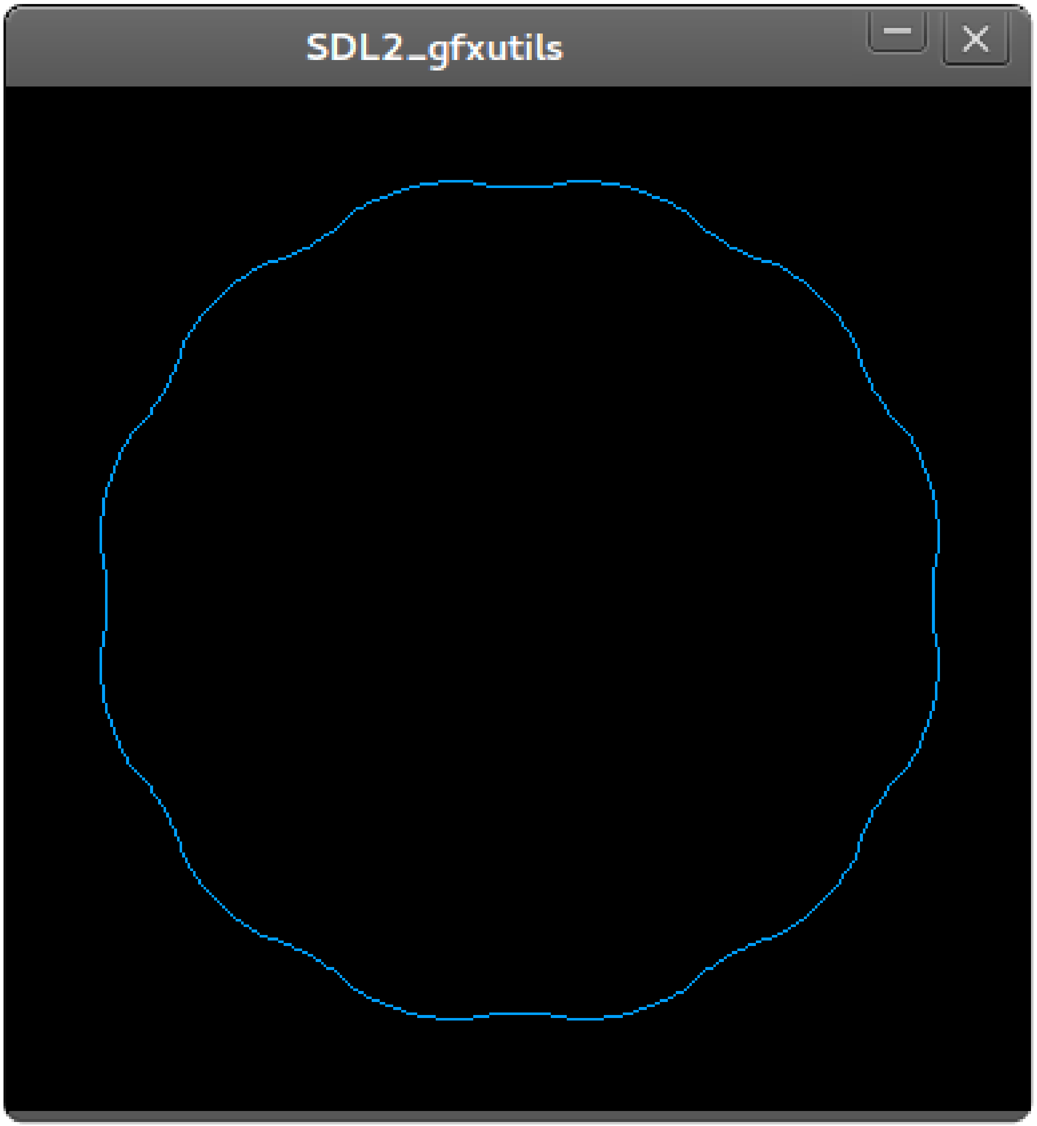


- display filled polygon

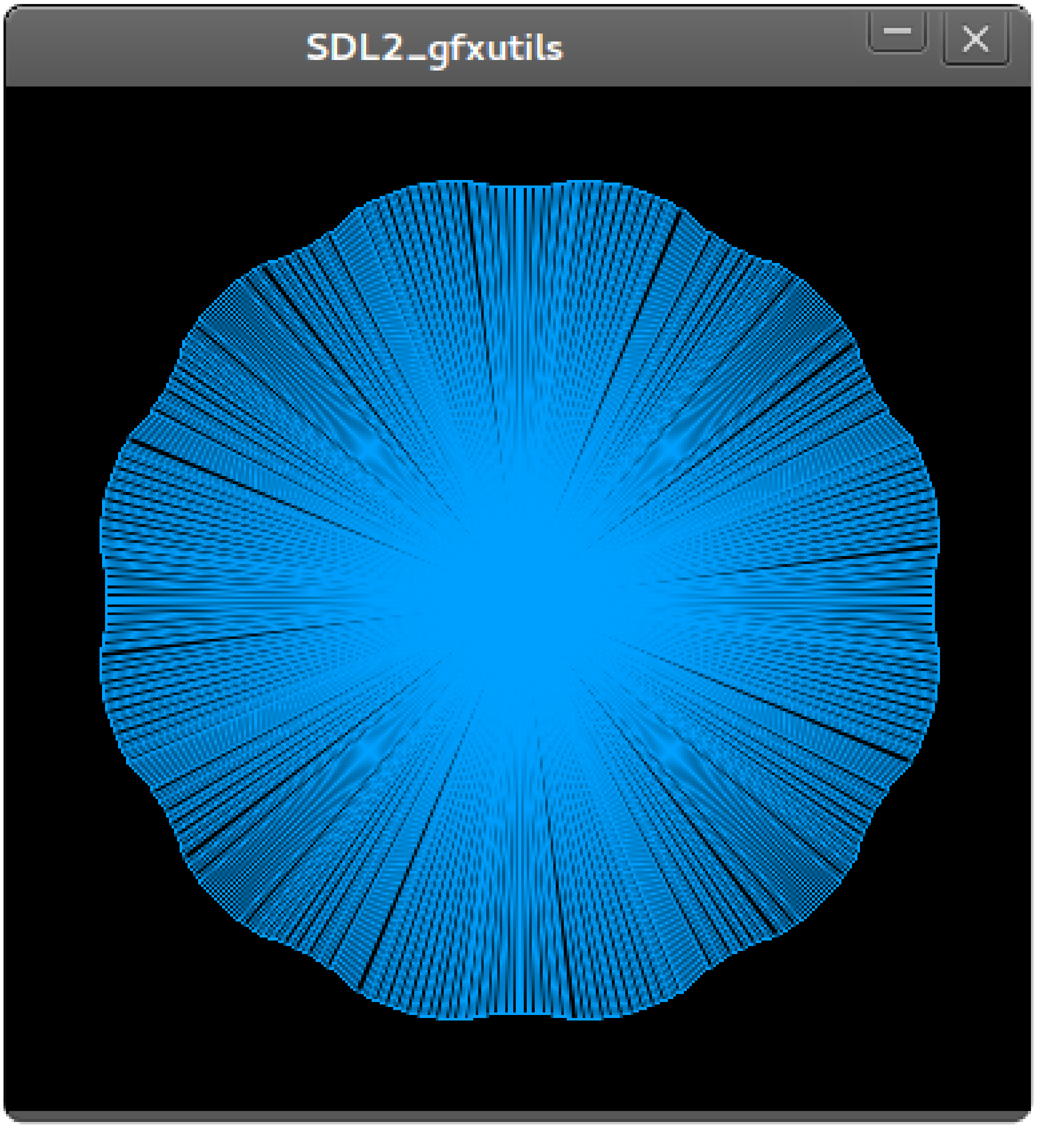


generate rounded inside out polygon

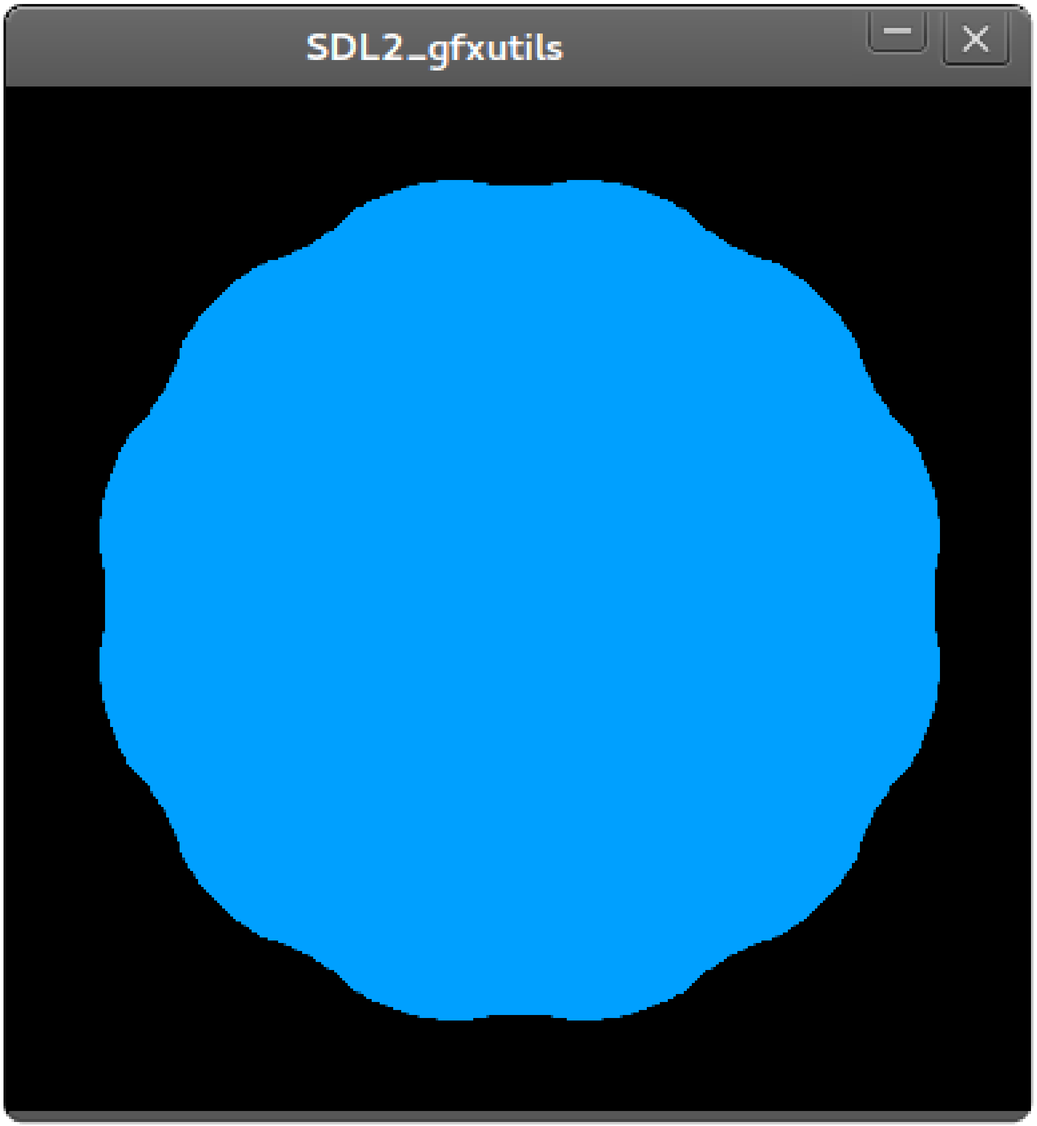
- display polygon



- display strikethrough polygon

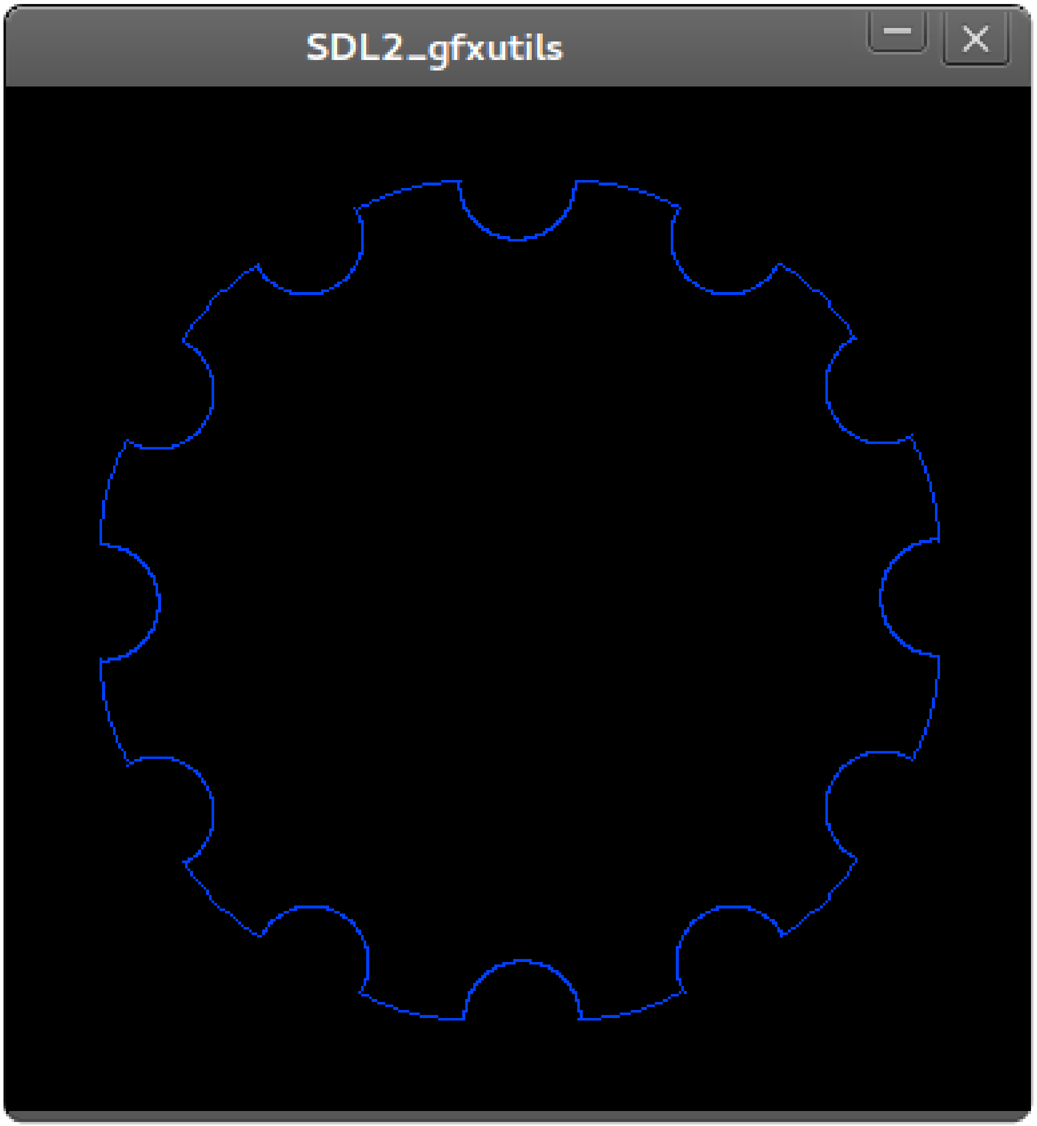


- display filled polygon

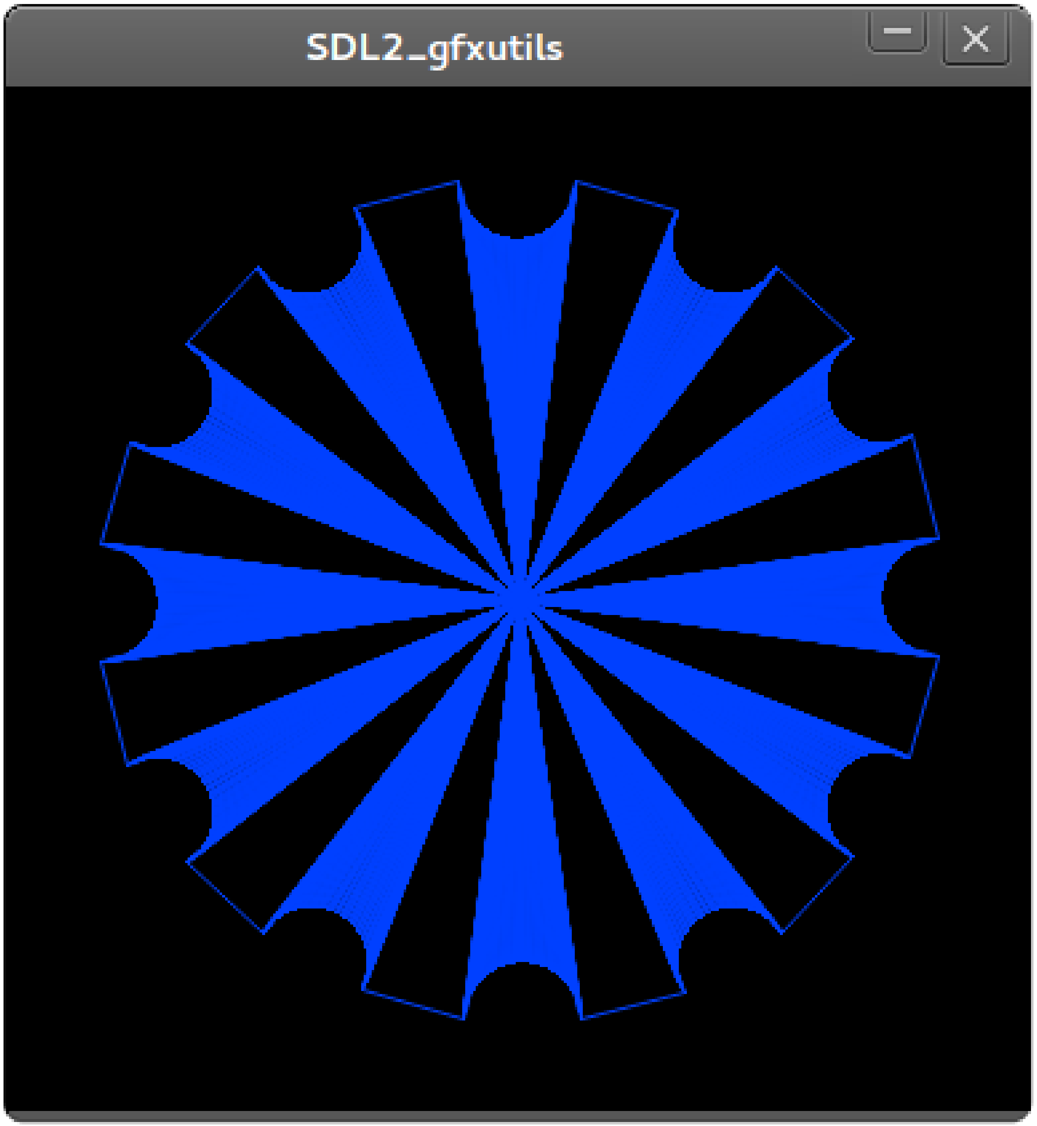


generate alternate inside half circle polygon

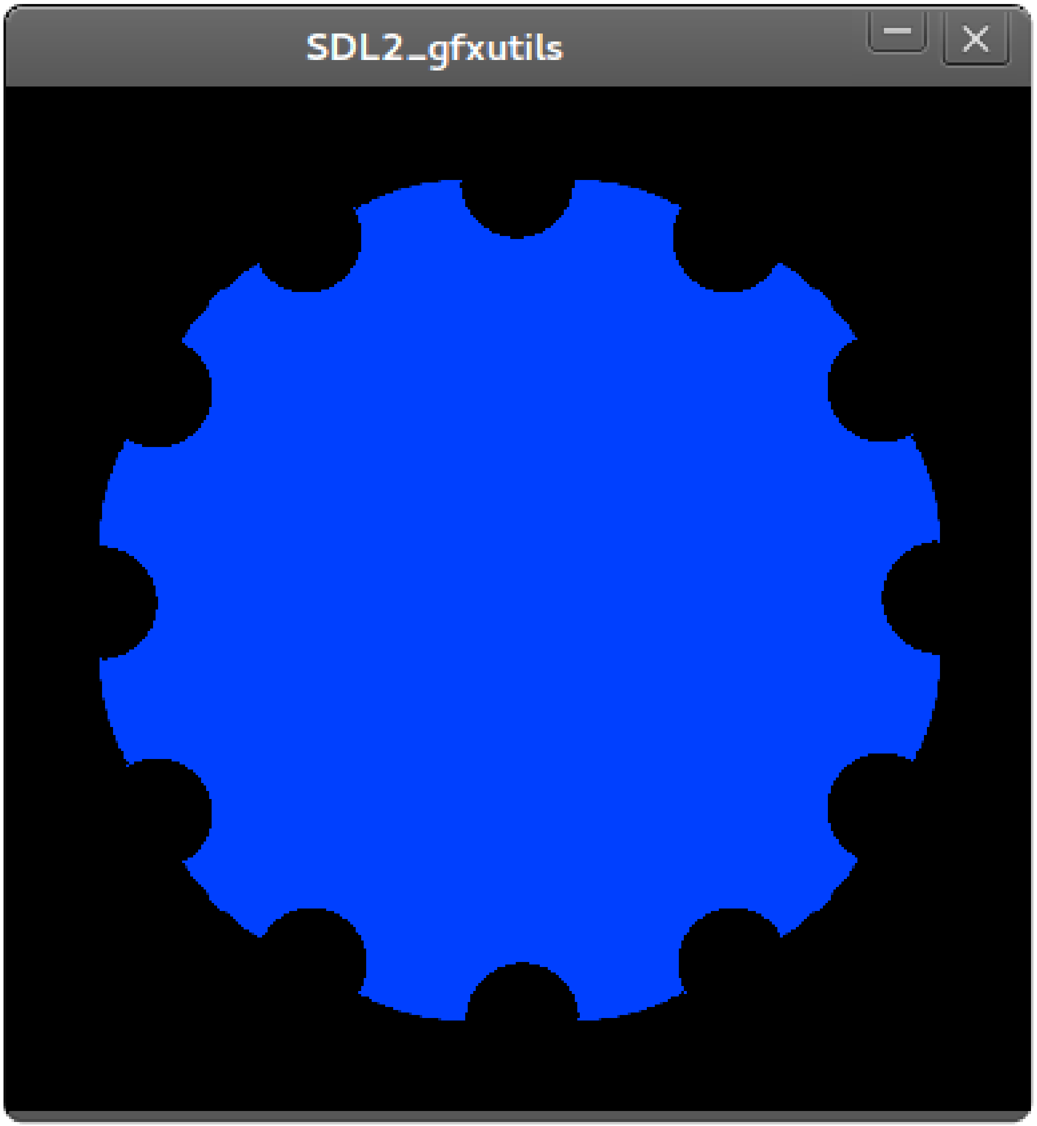
- display polygon



- display strikethrough polygon

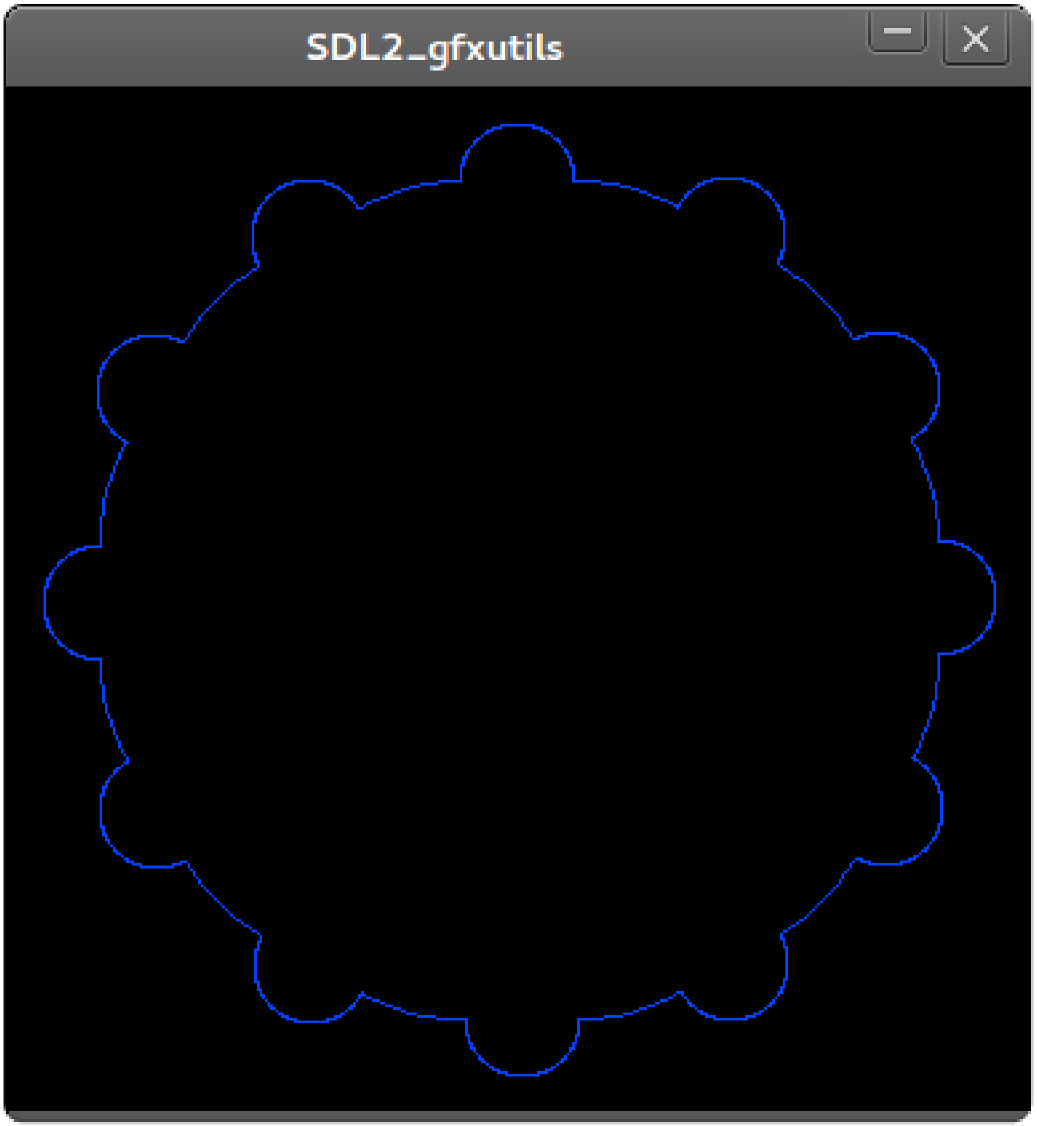


- display filled polygon

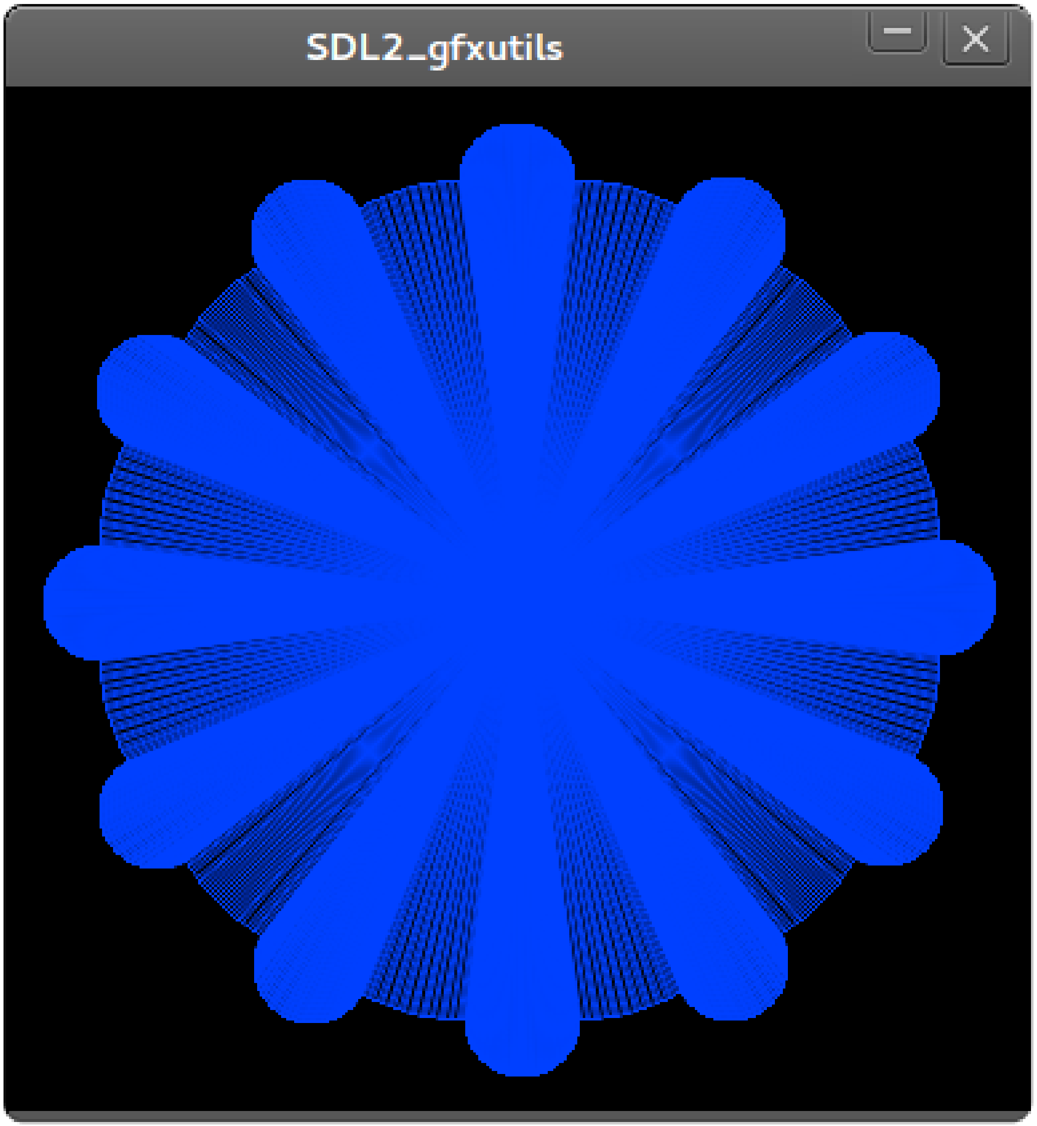


generate alternate outside half circle polygon

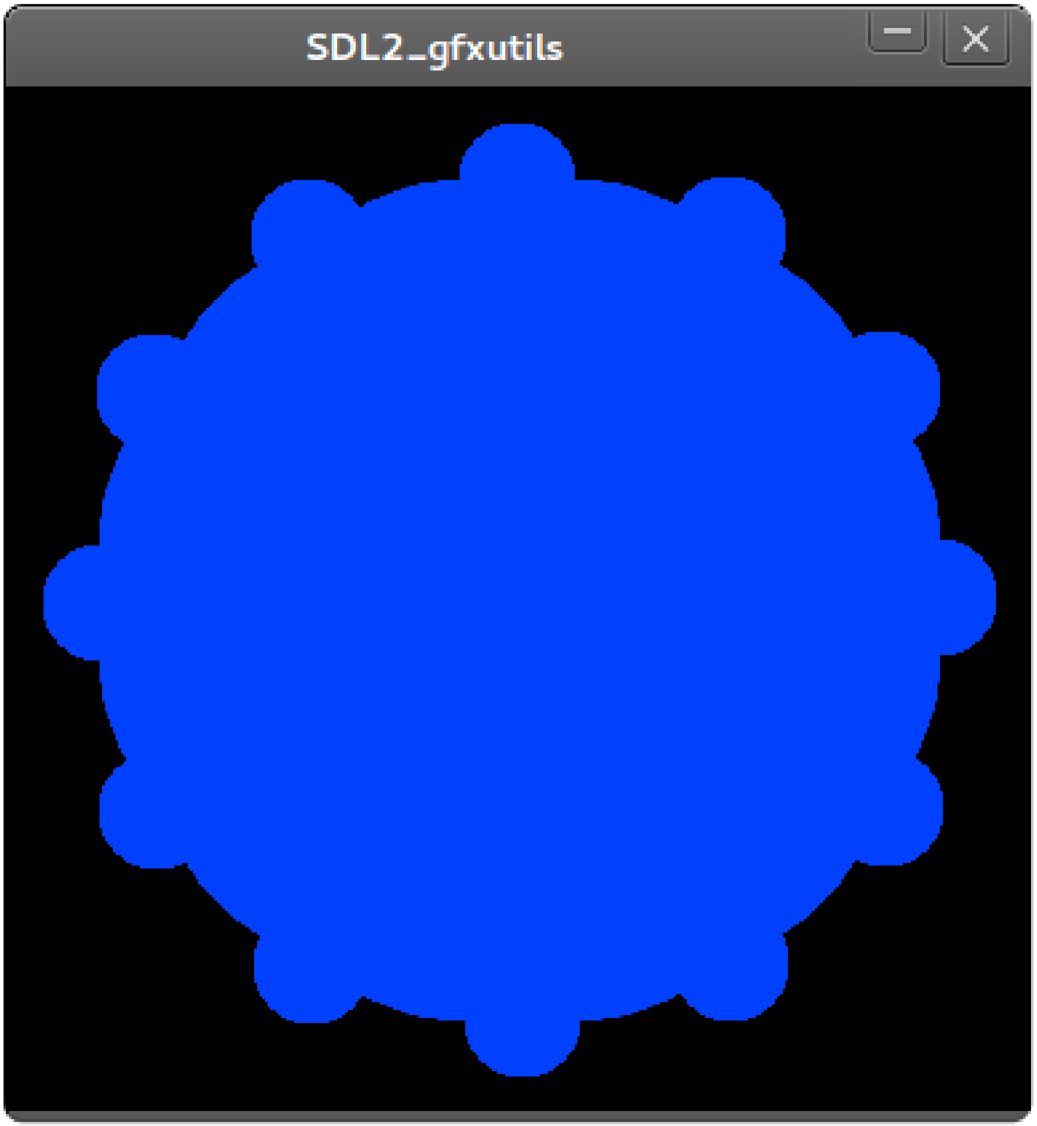
- display polygon



- display strikethrough polygon

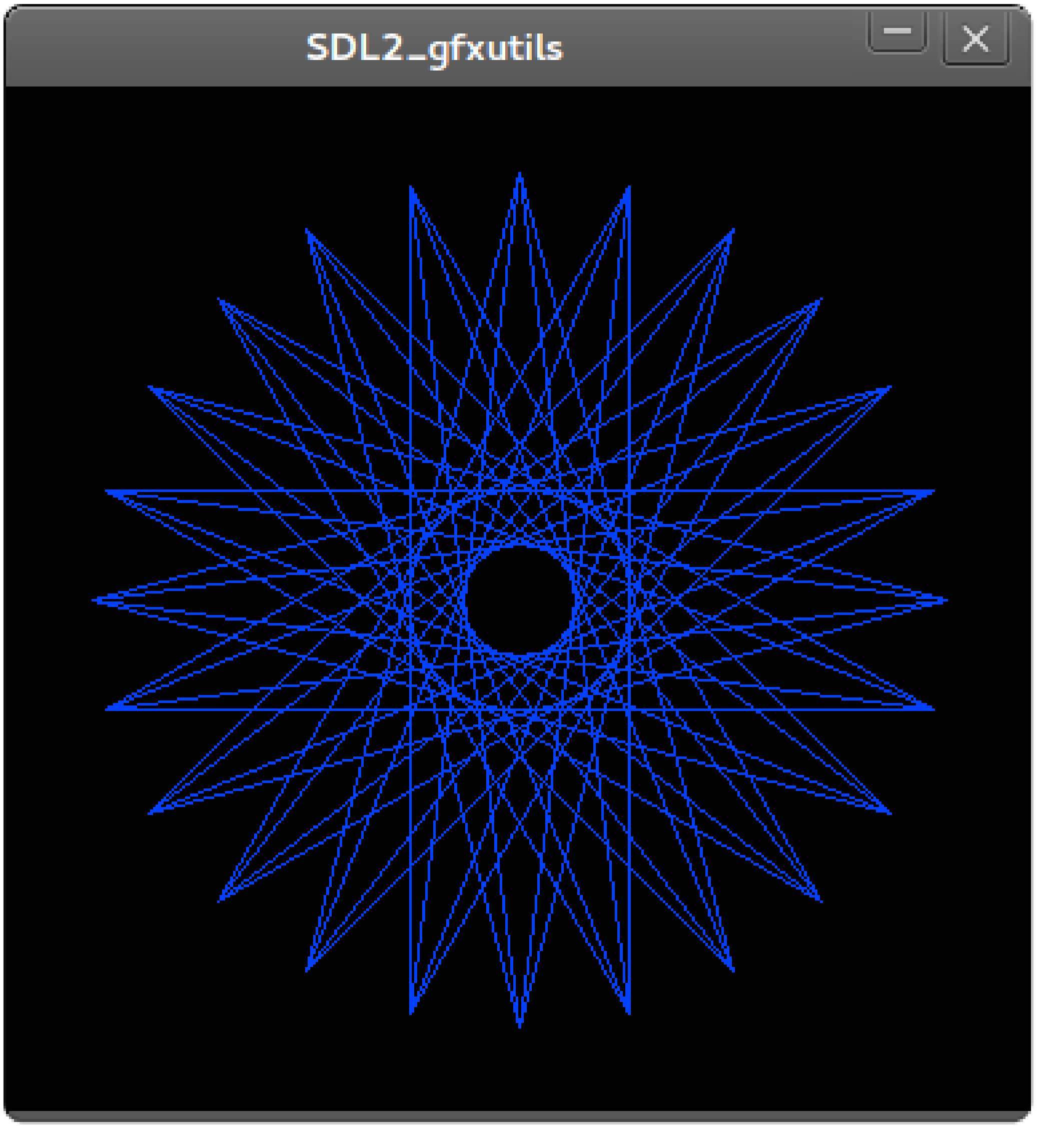


- display filled polygon

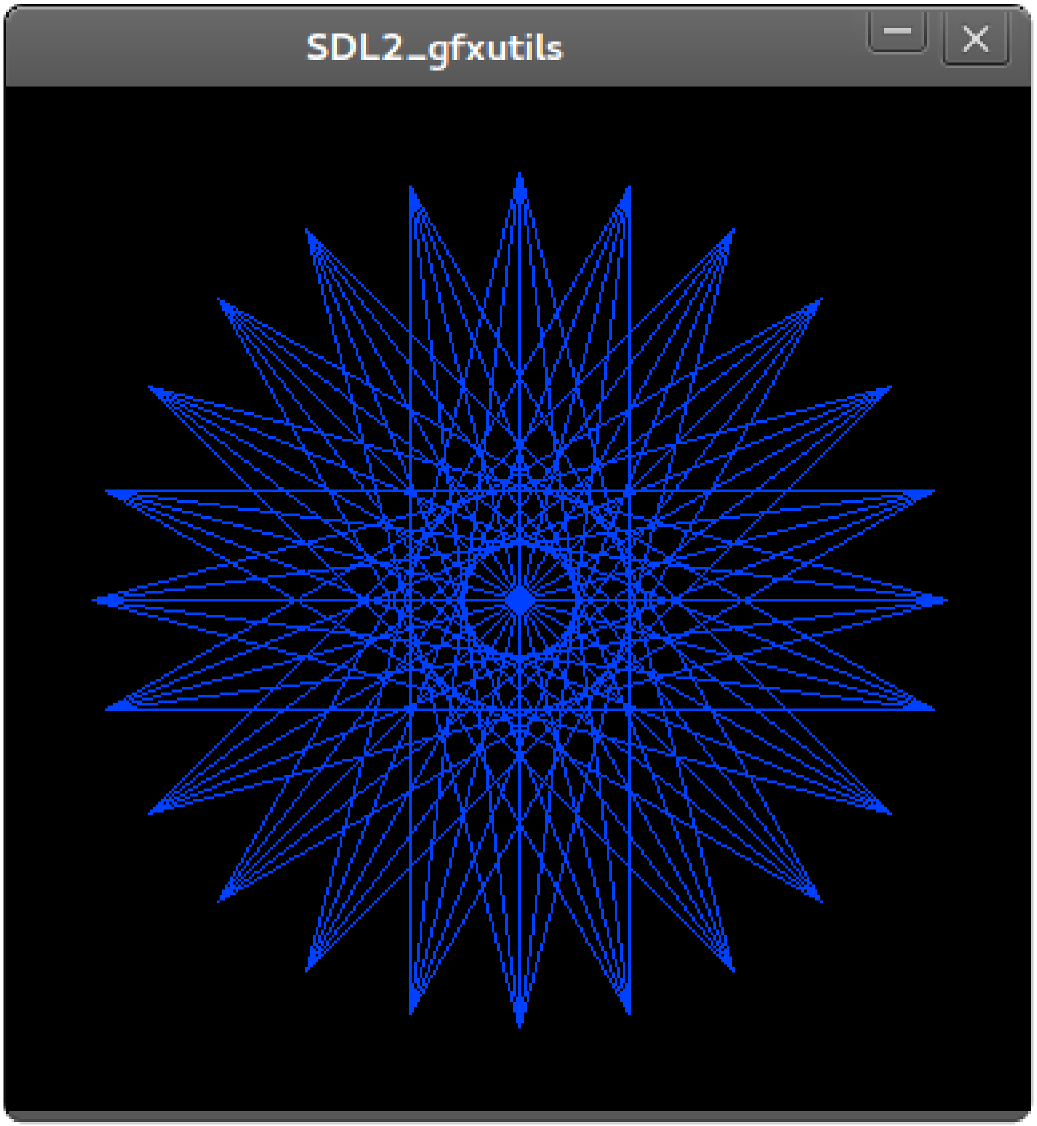


generate fractal

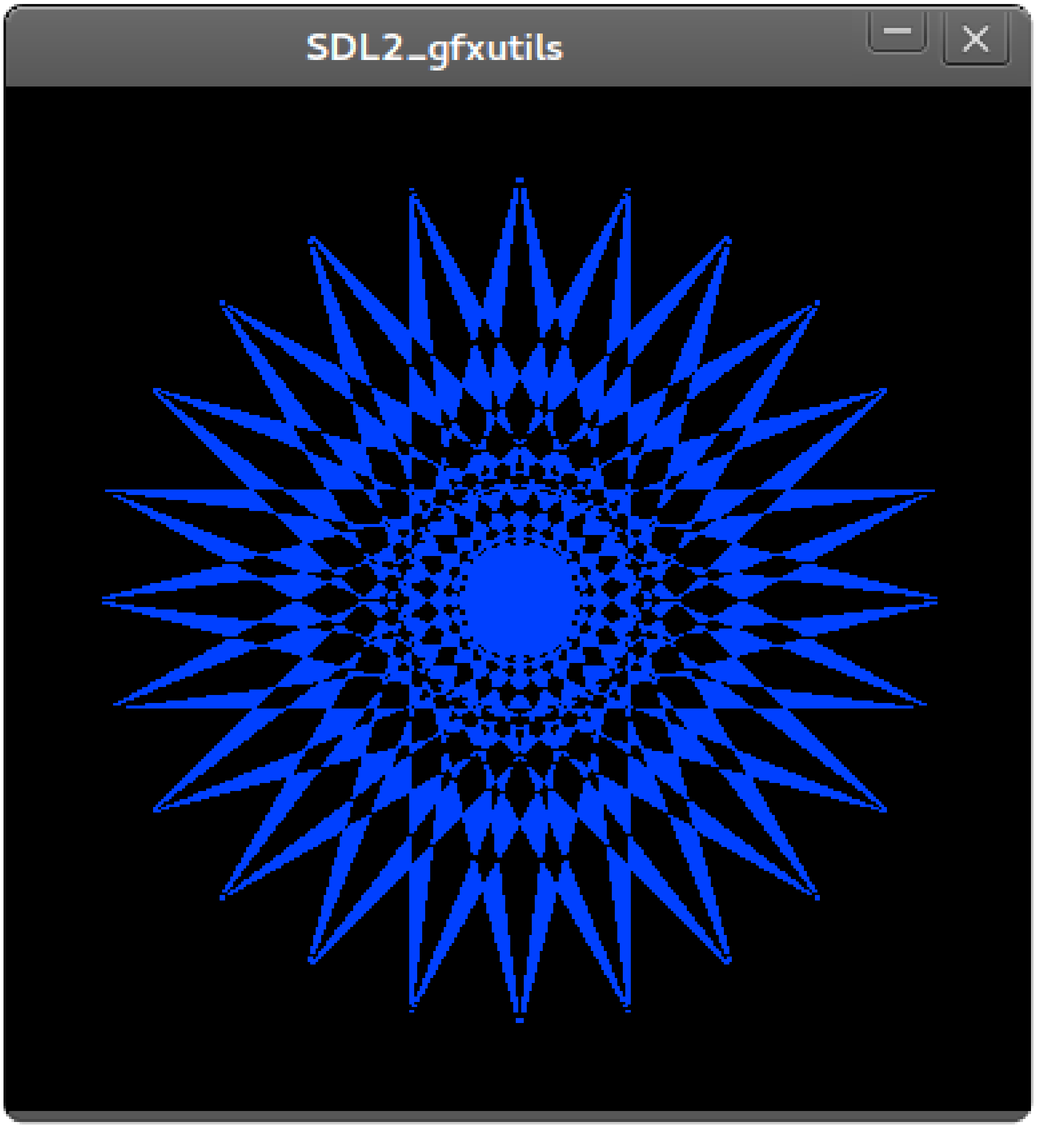
- display polygon



- display strikethrough polygon

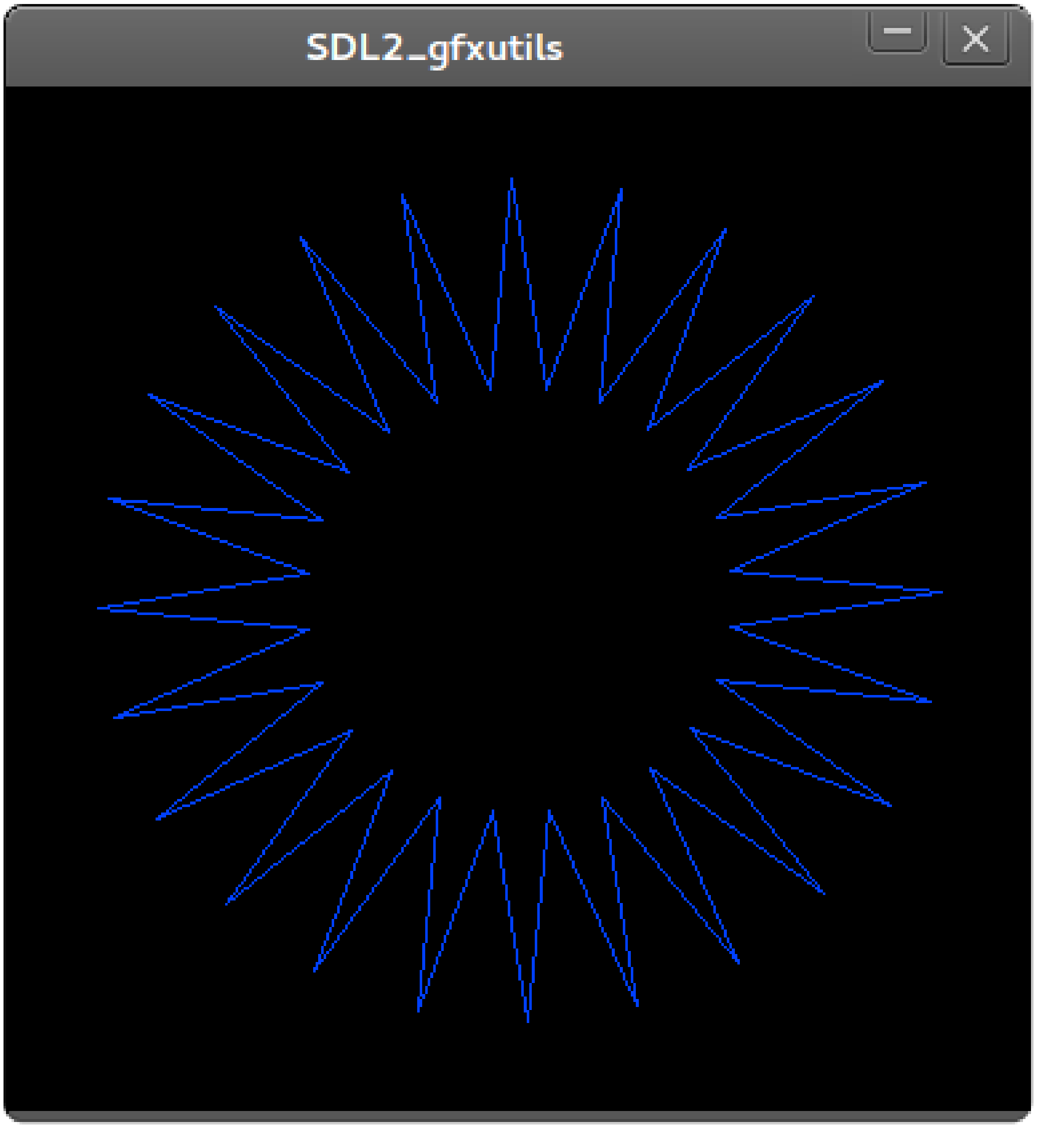


- display filled polygon

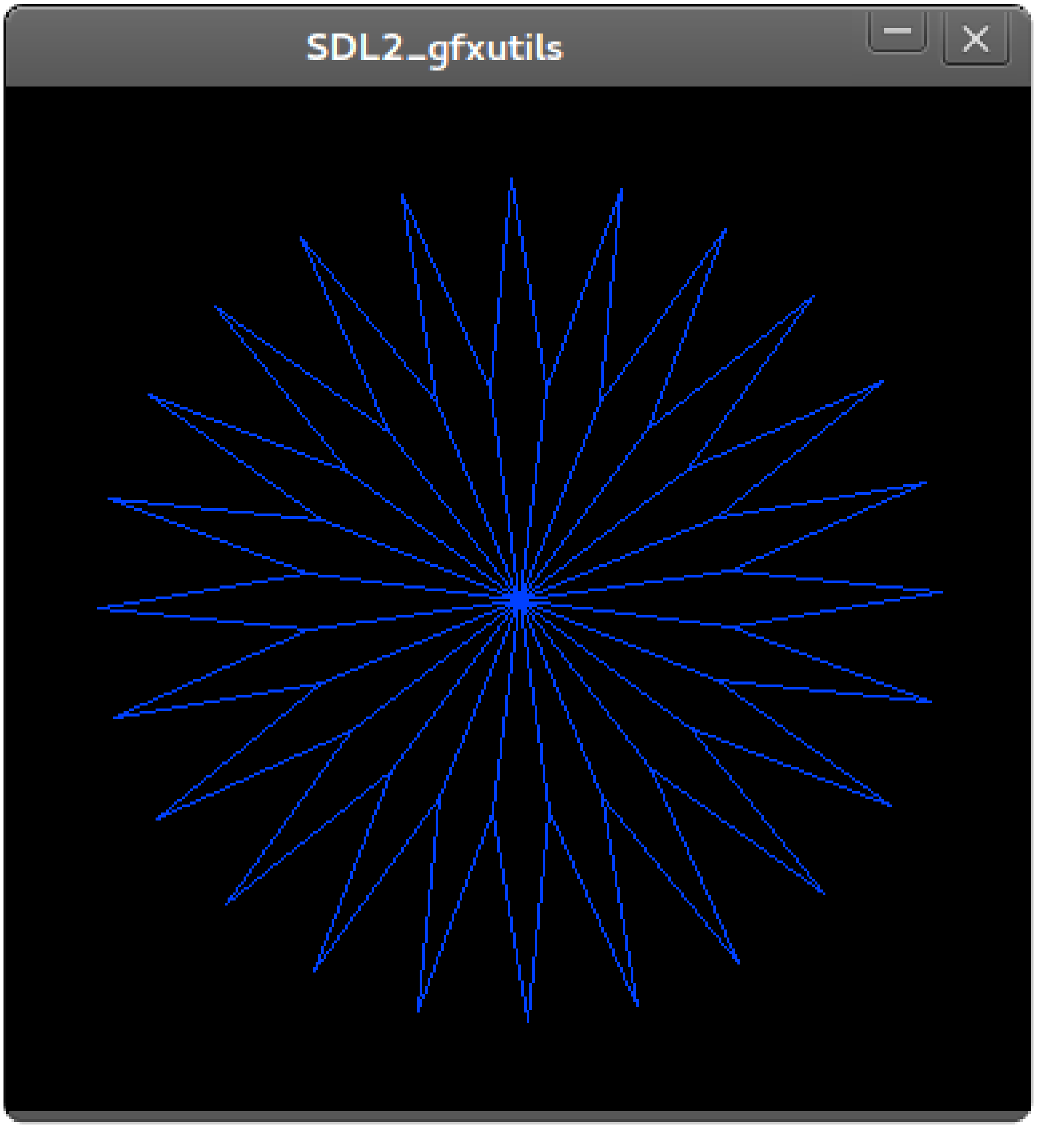


generate star

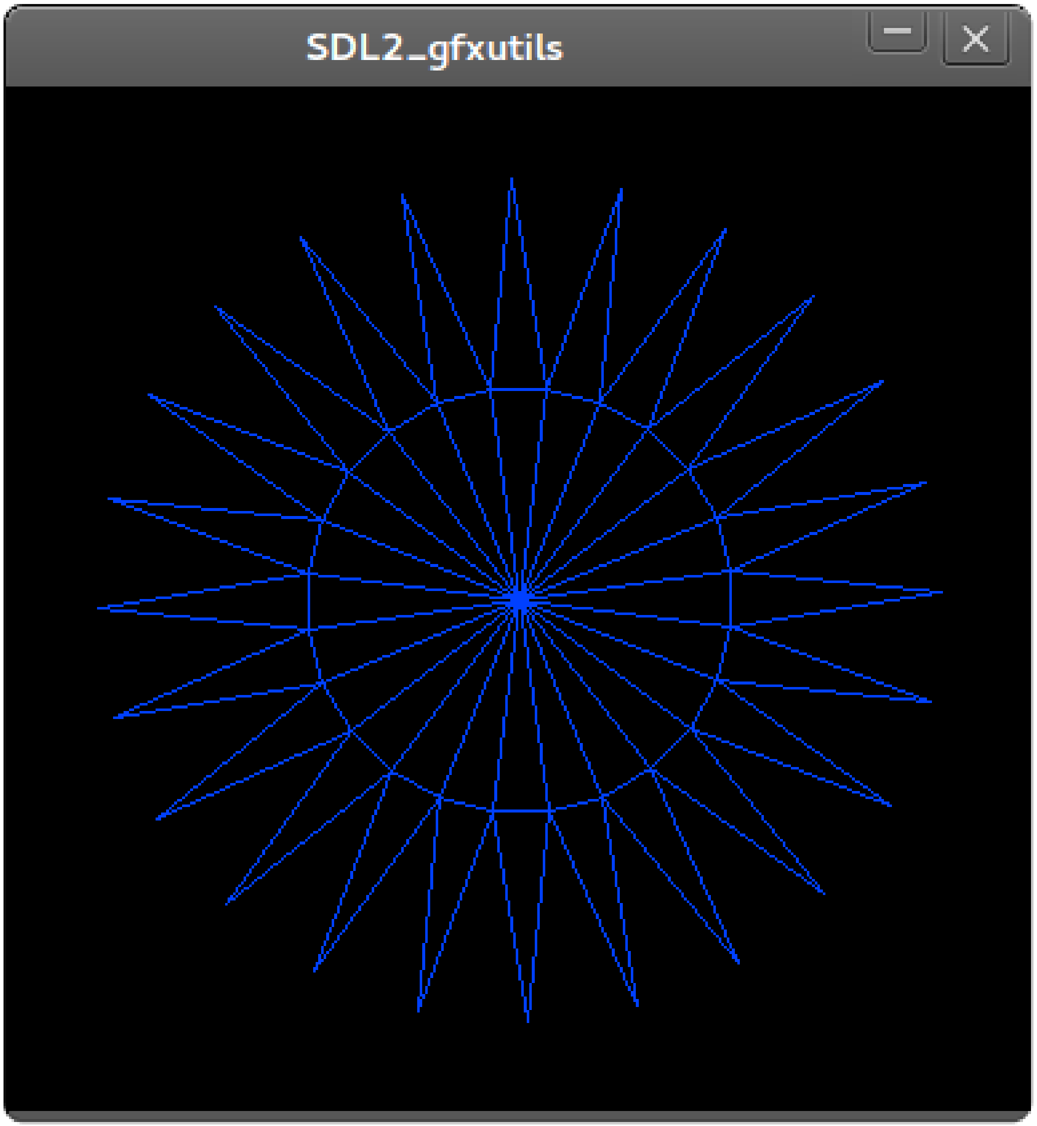
- display star



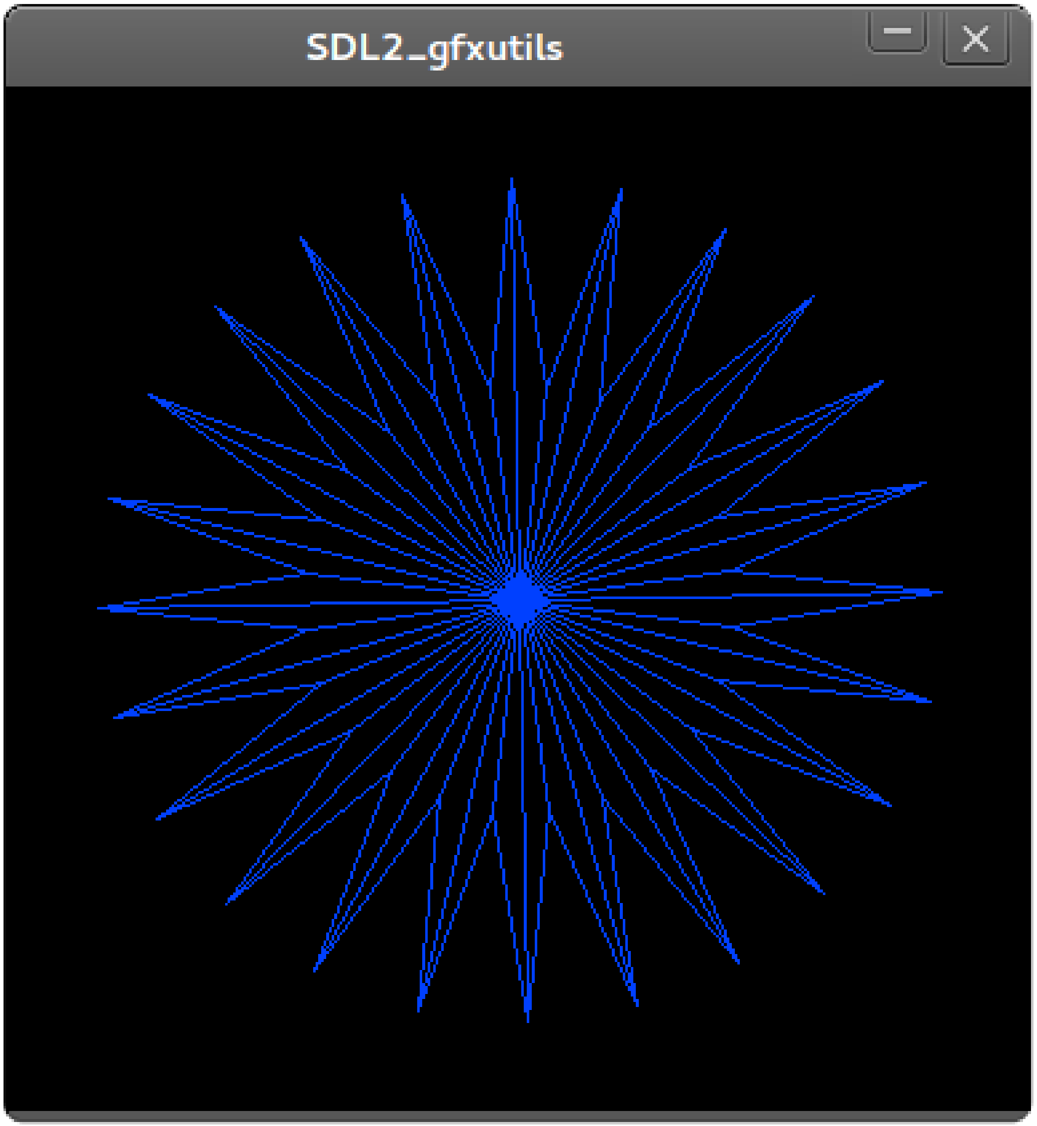
- display flower star



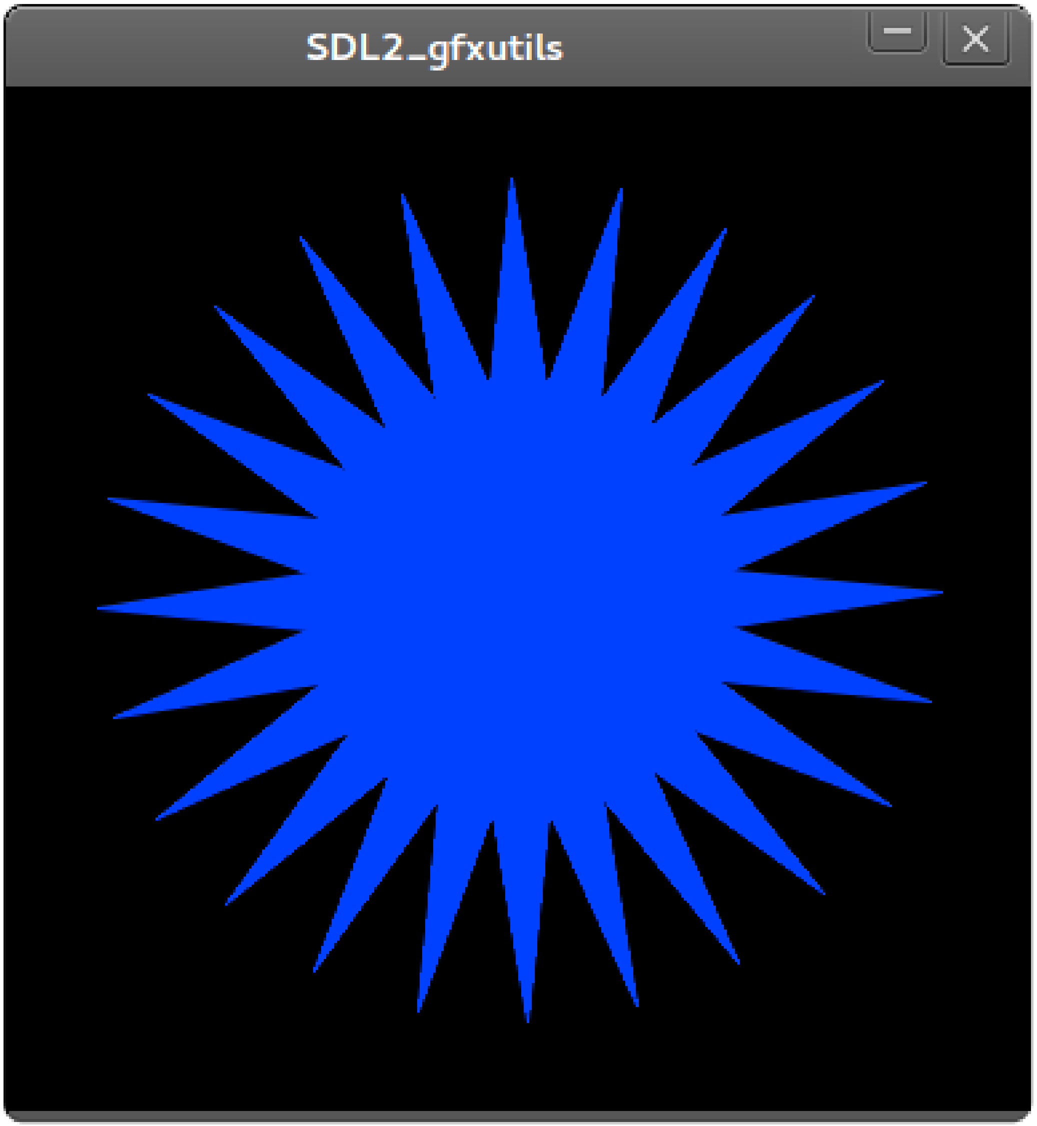
- display polygon star



- display strikethrough star

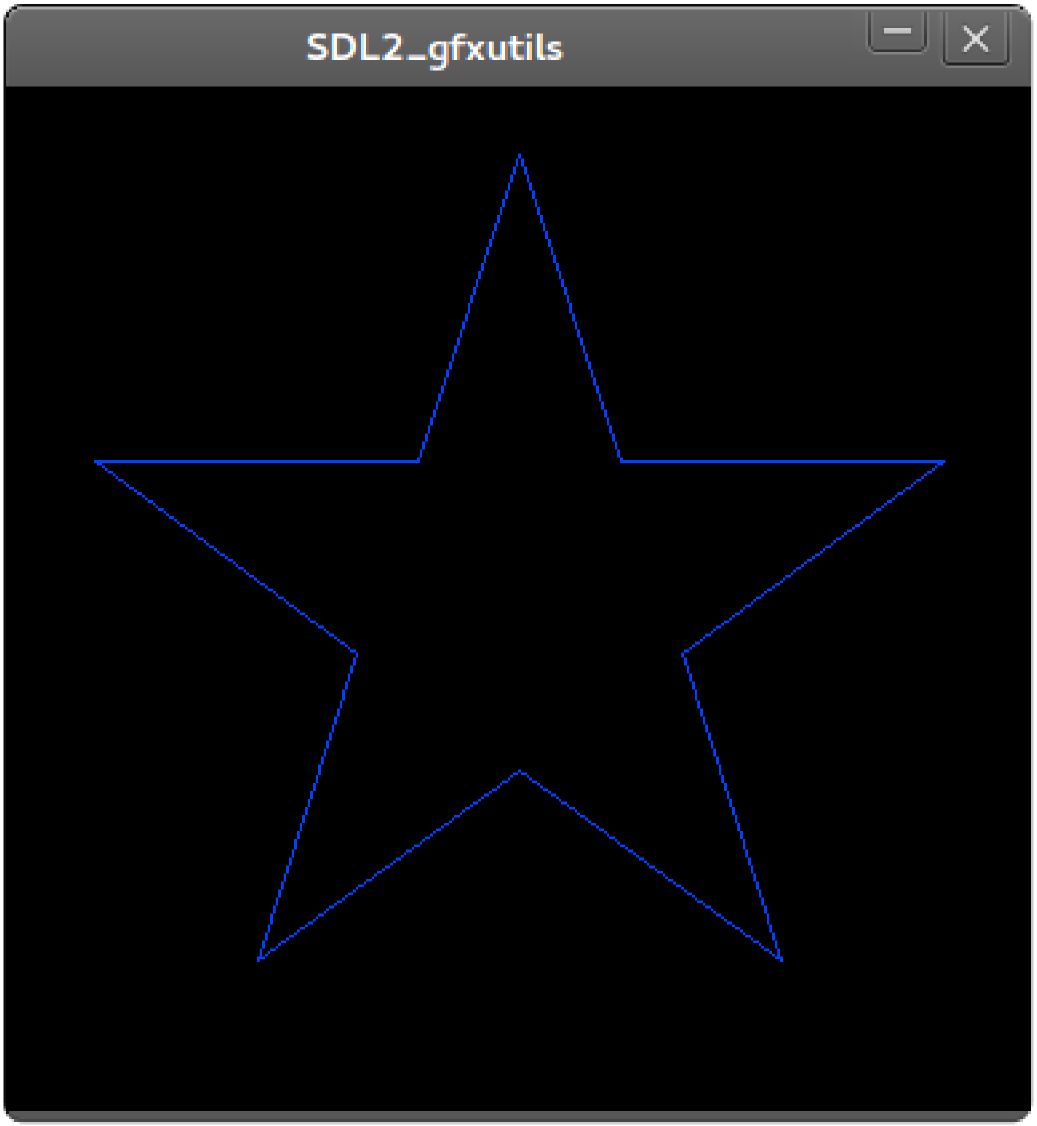


- display filled star

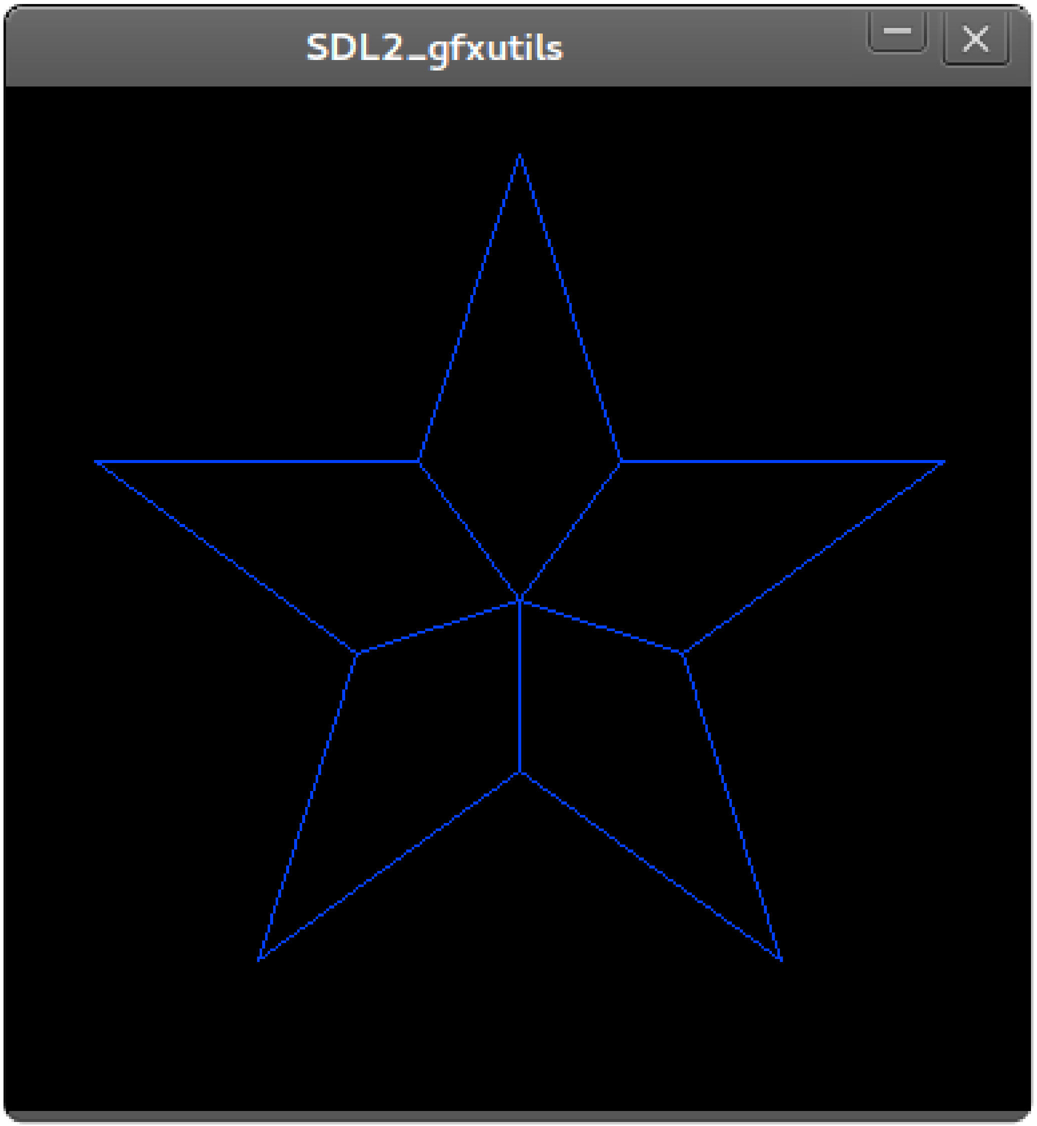


generate pentagram star

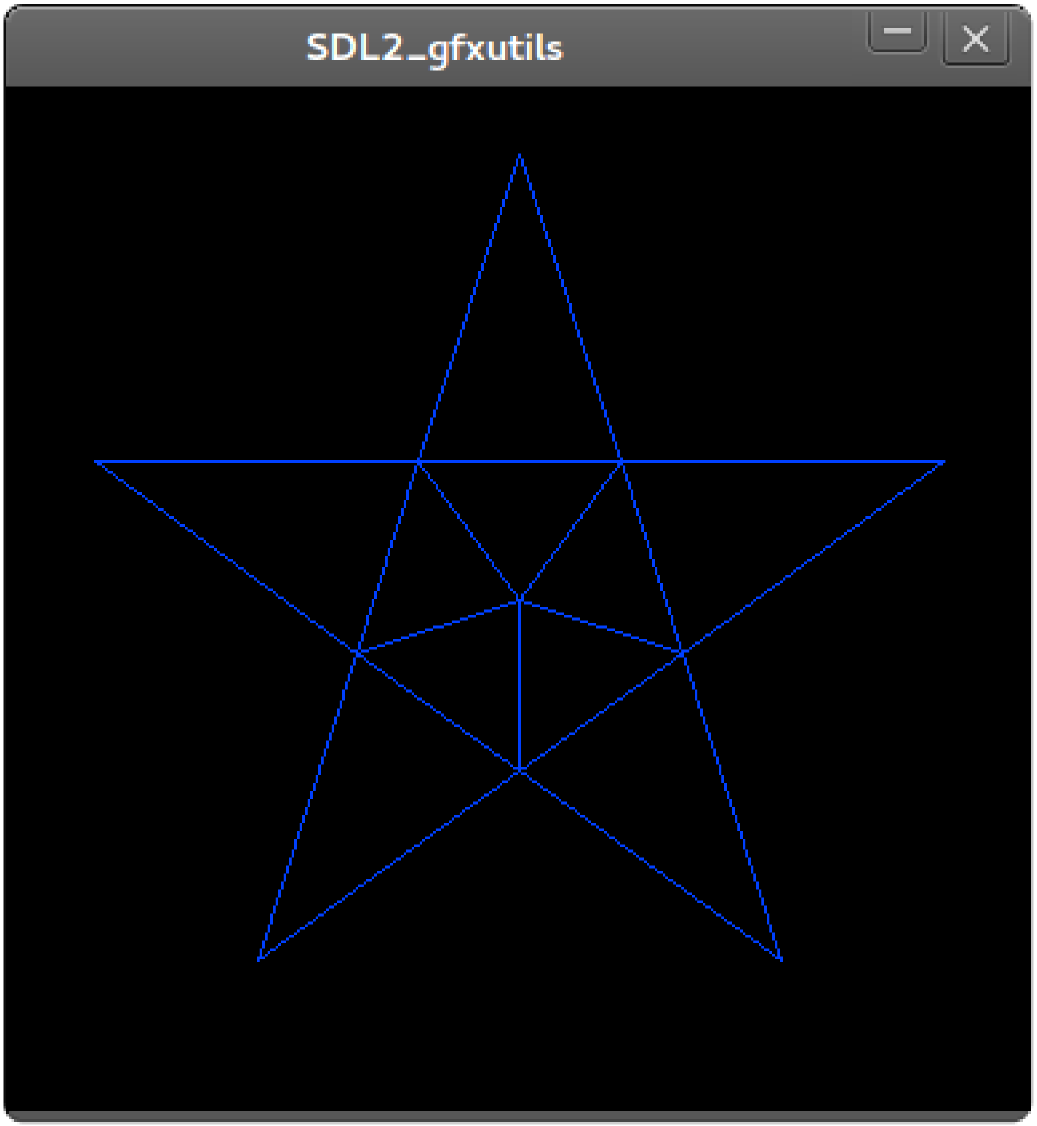
- display star



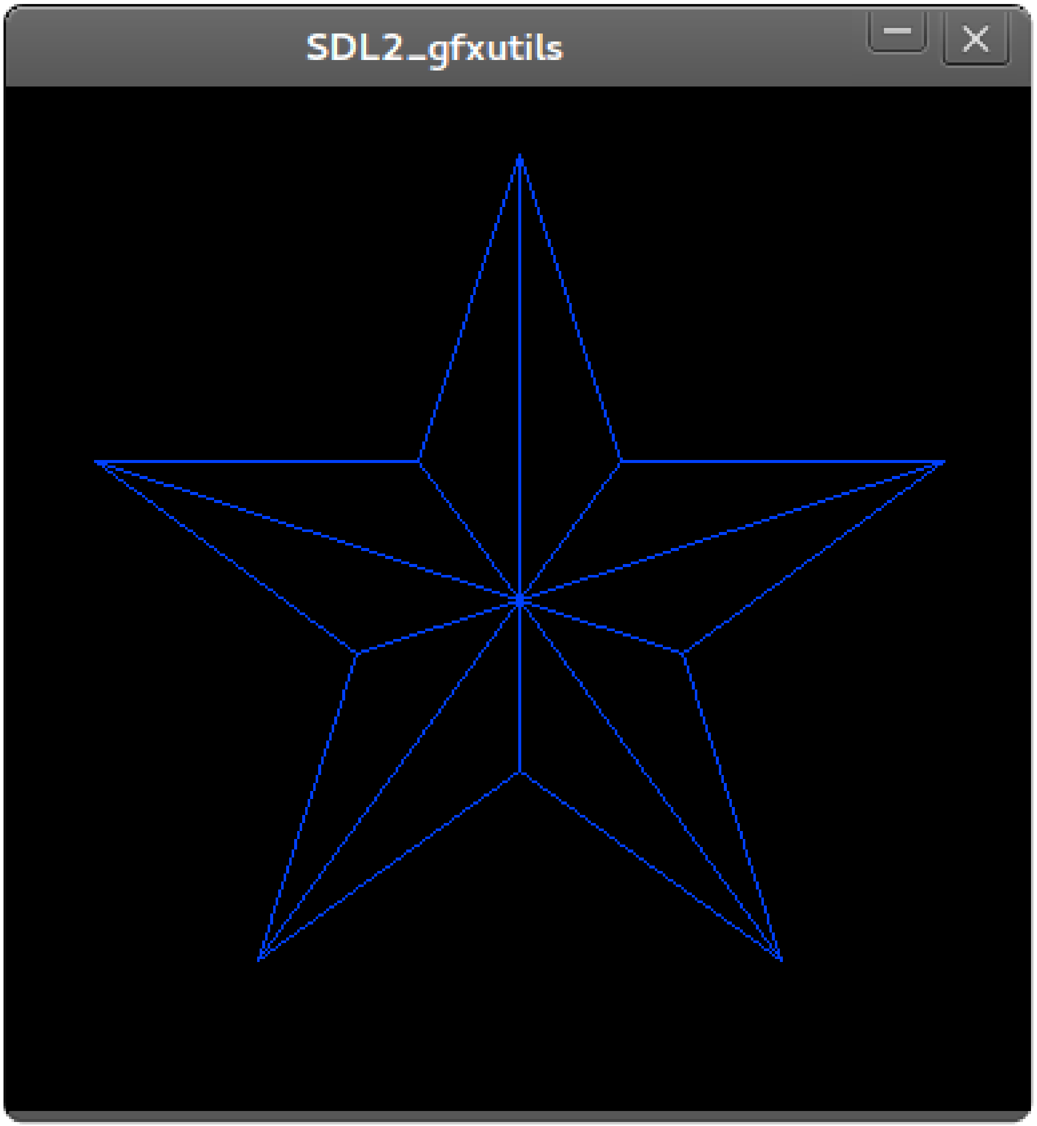
- display flower star



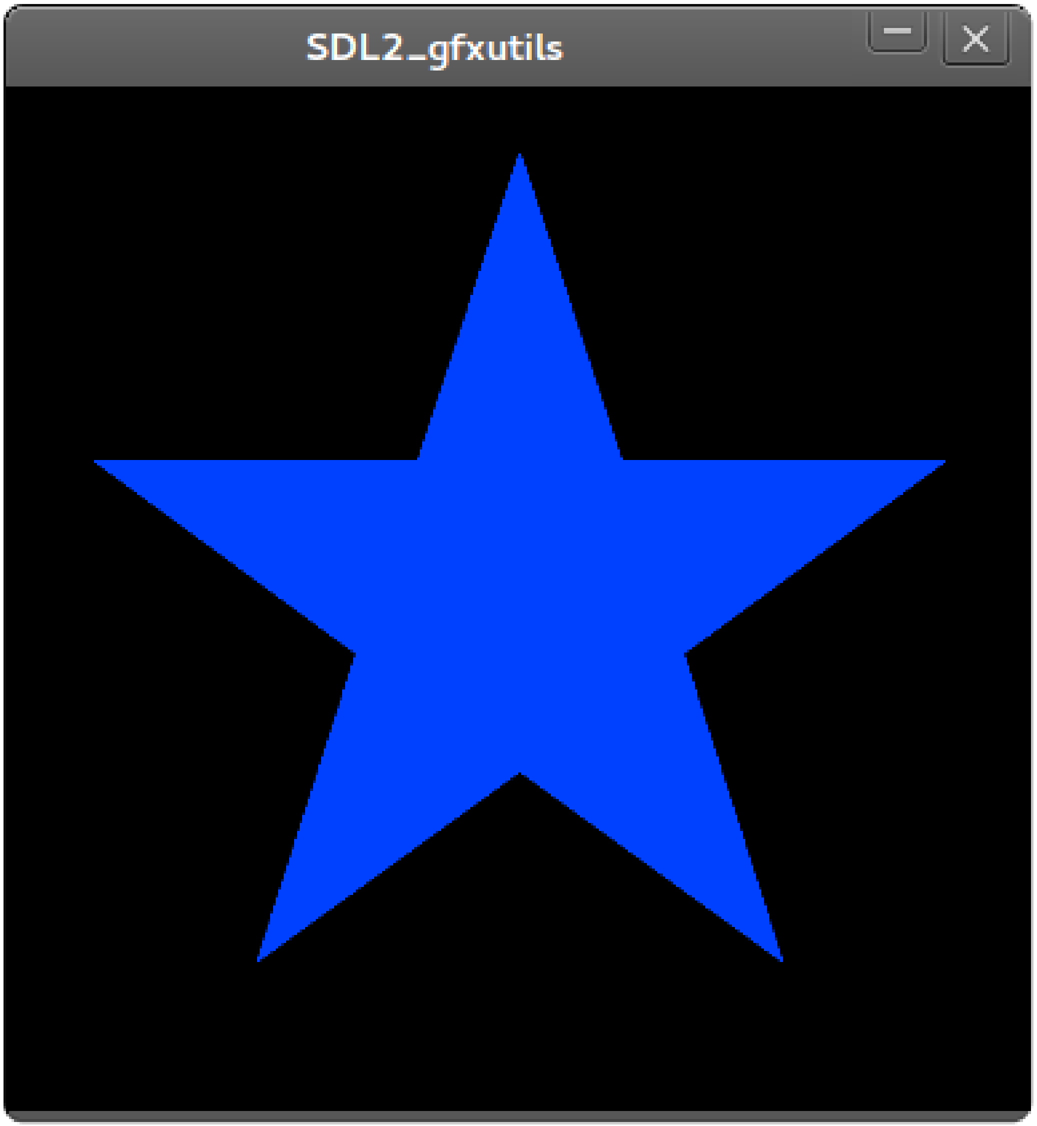
- display polygon star



- display strikethrough star

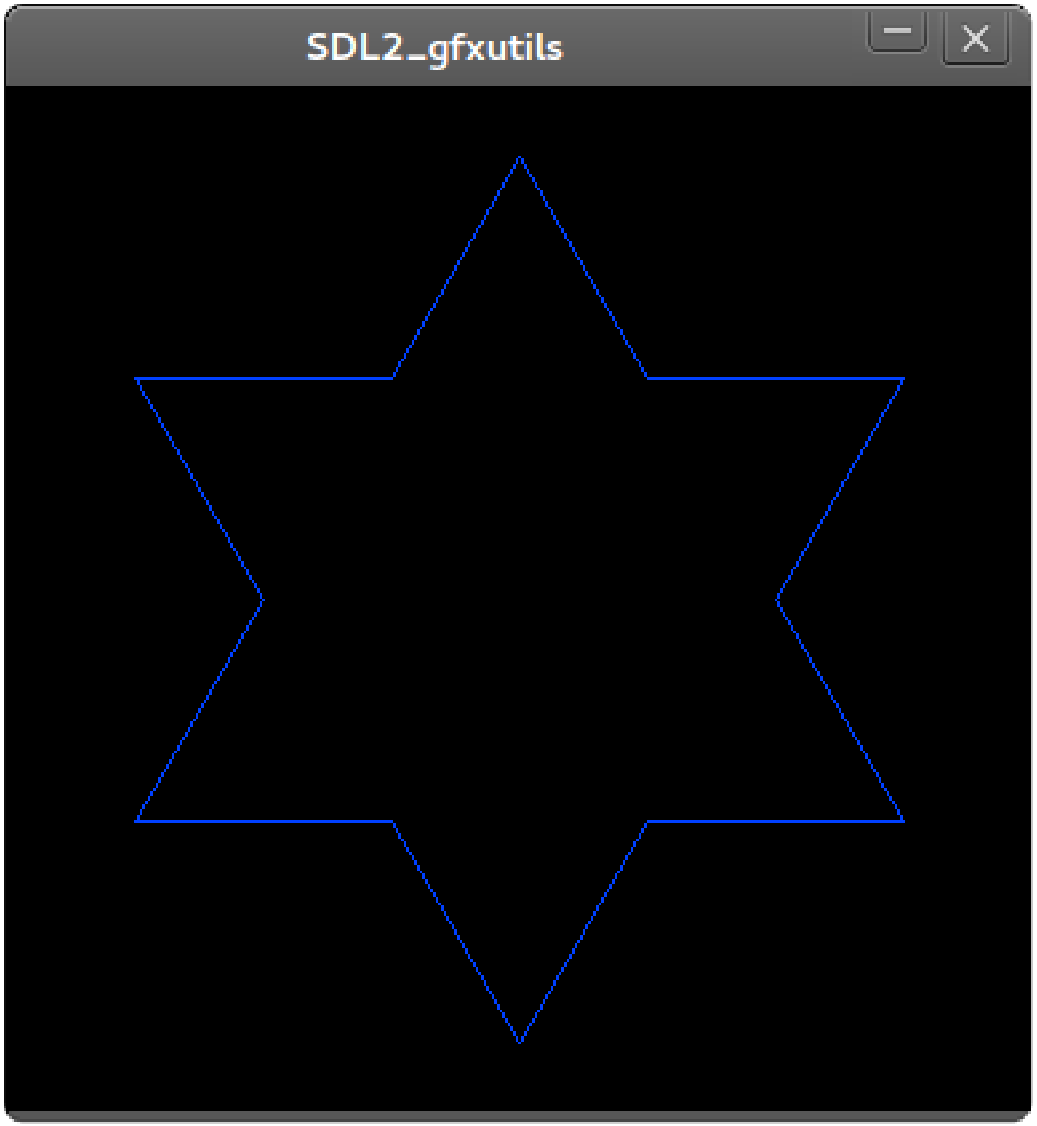


- display filled star

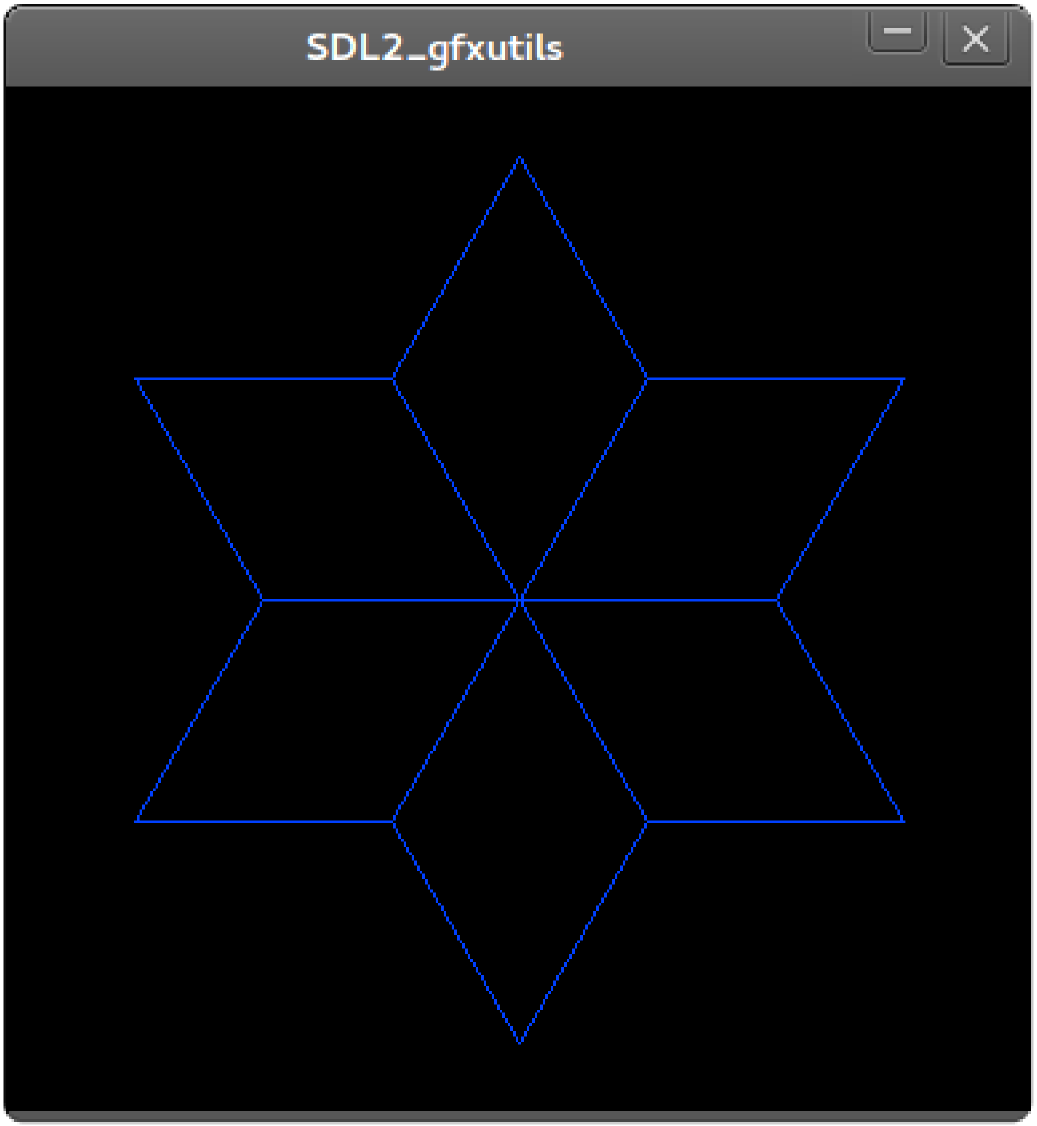


generate hexagram star

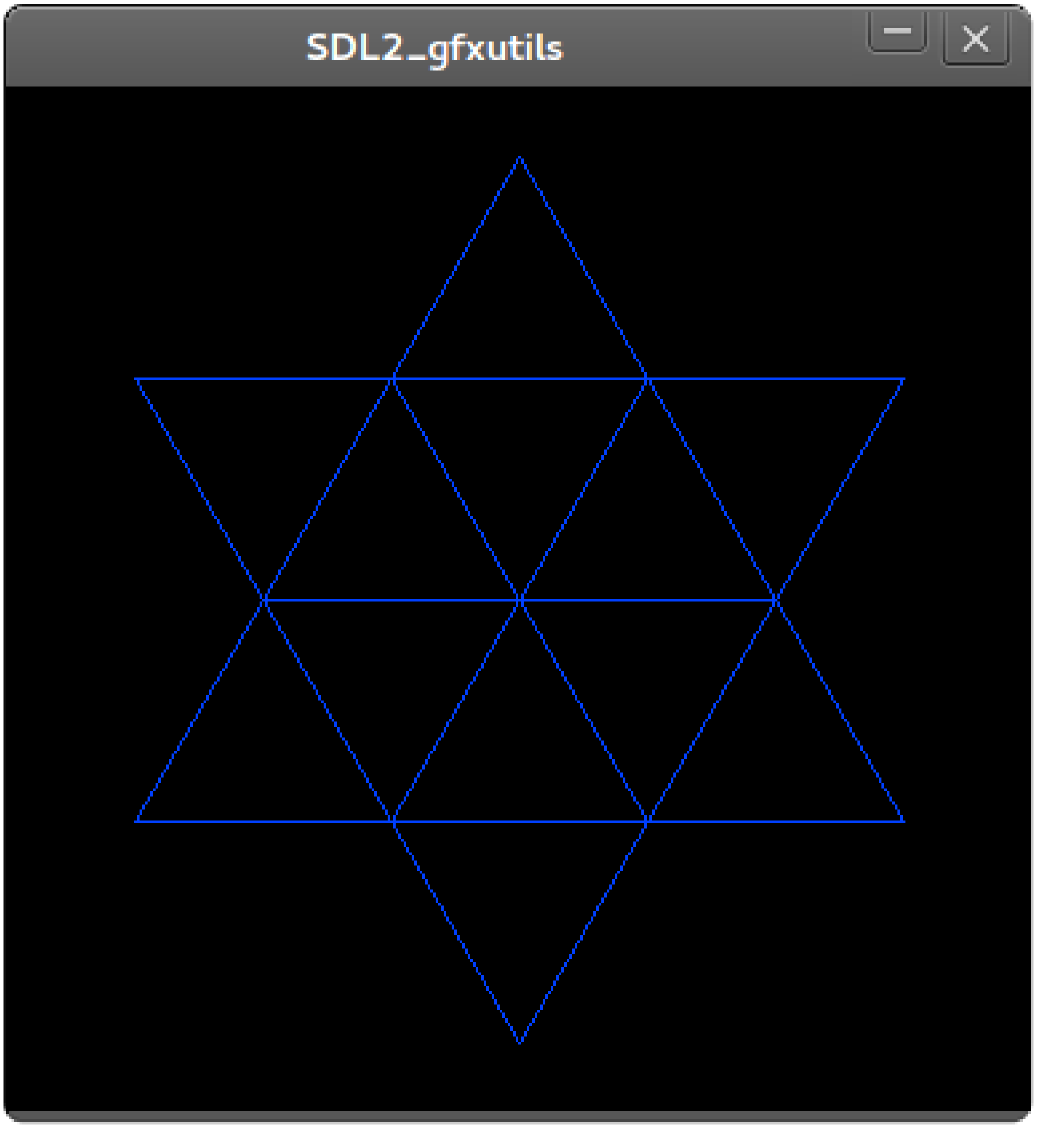
- display star



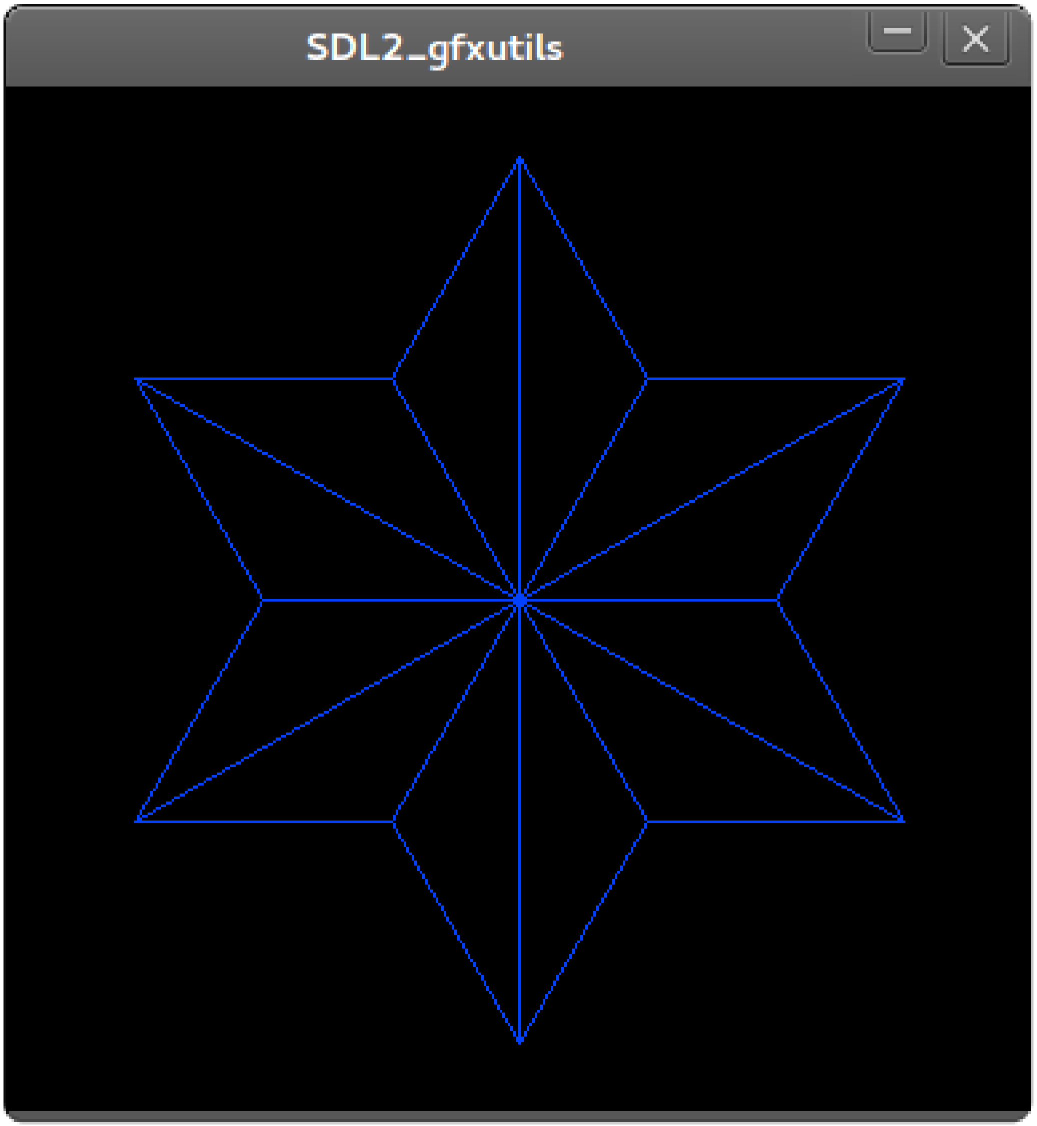
- display flower star



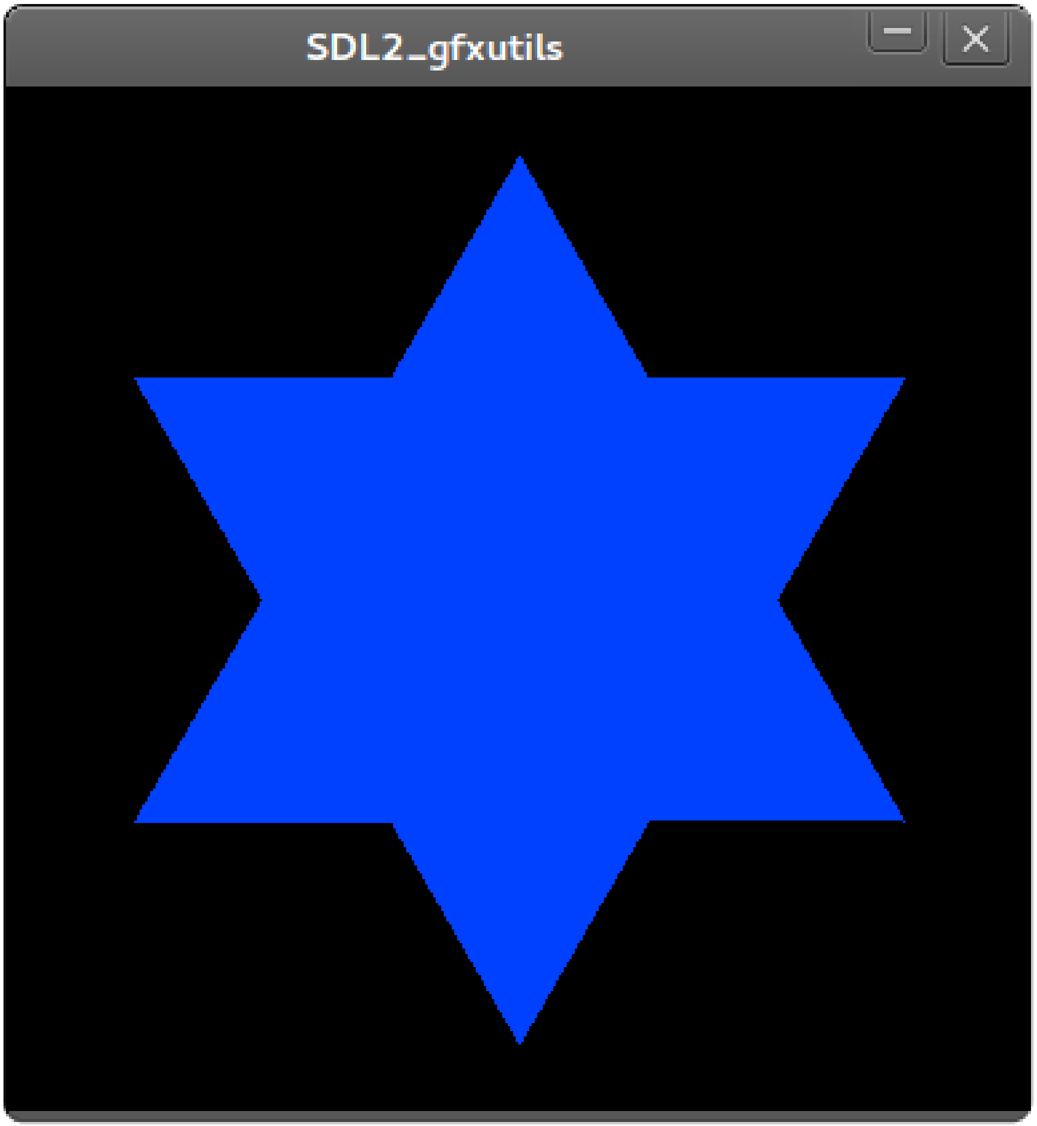
- display polygon star



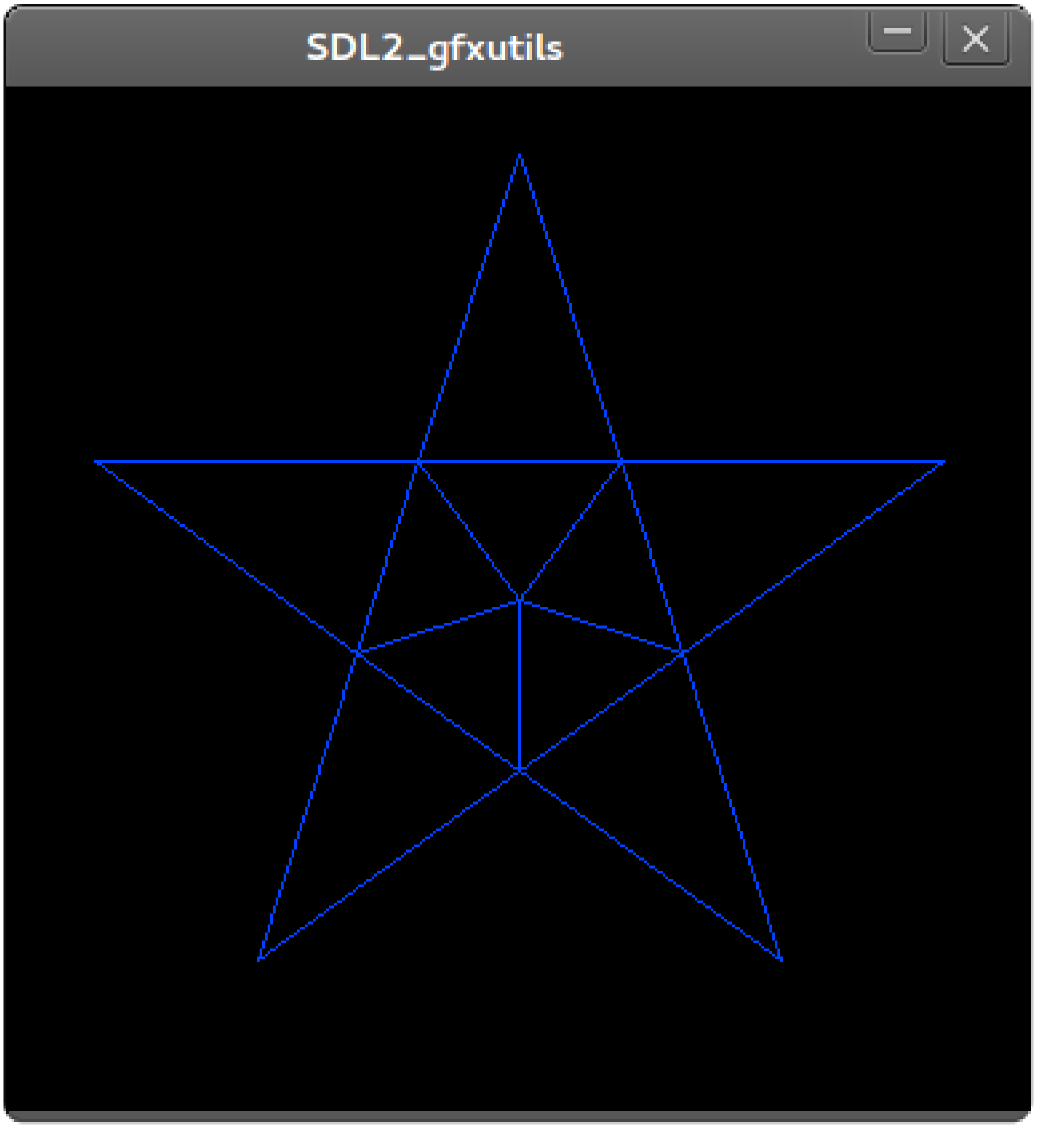
- display strikethrough star



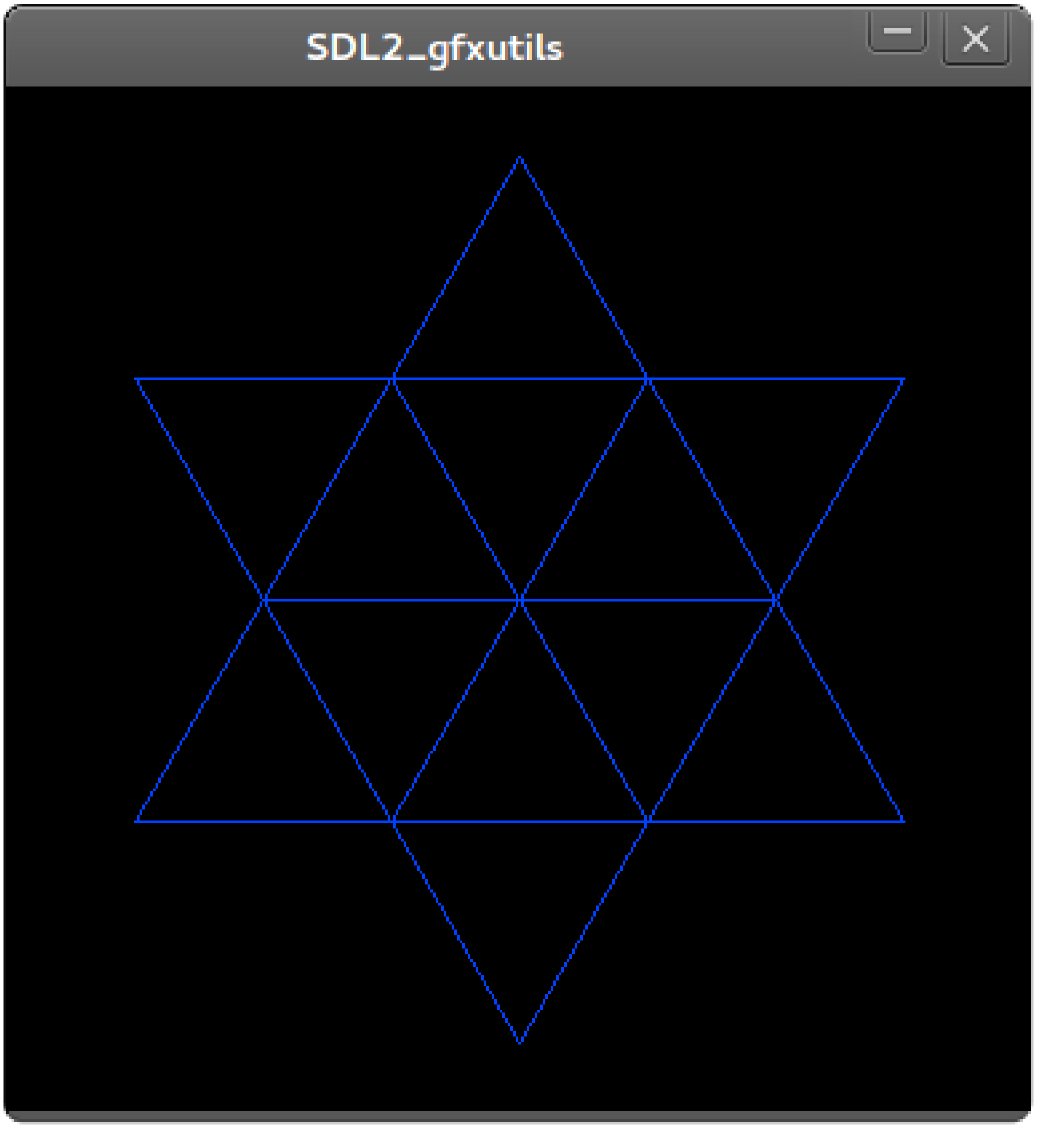
- display filled star


SDL2_gfxutils

generate pentagram

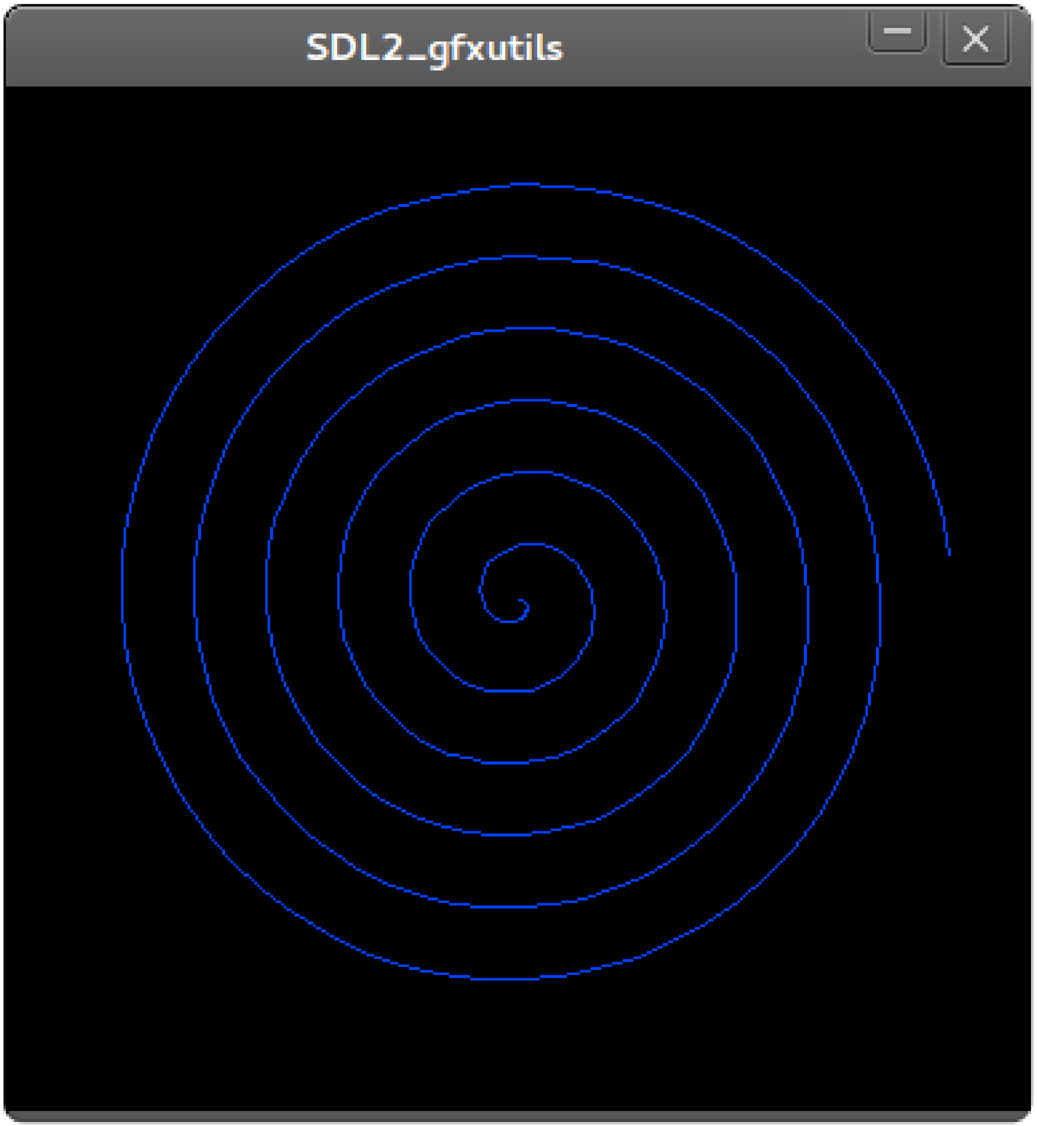


generate hexagram



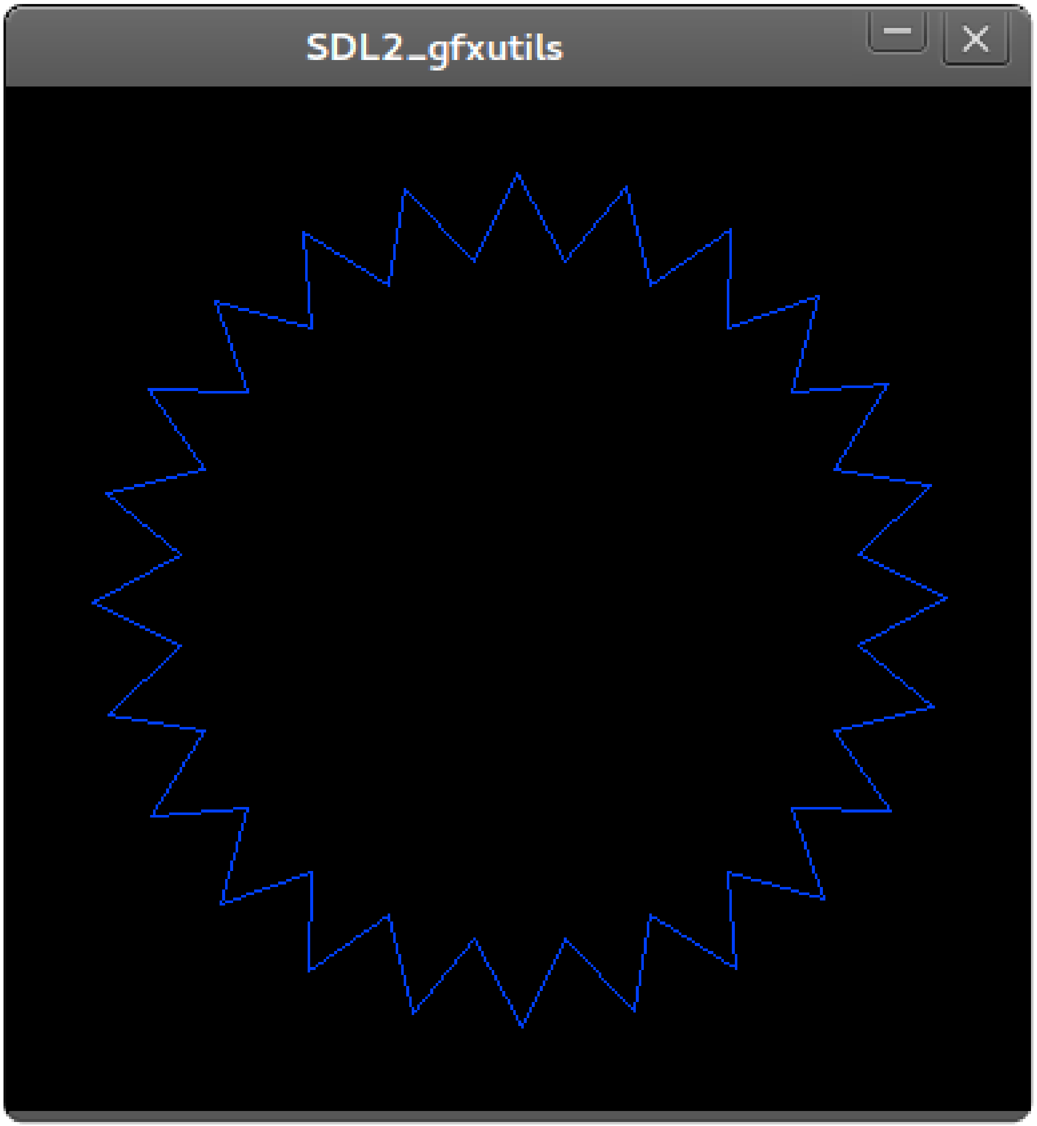
generate spiral

- display spiral

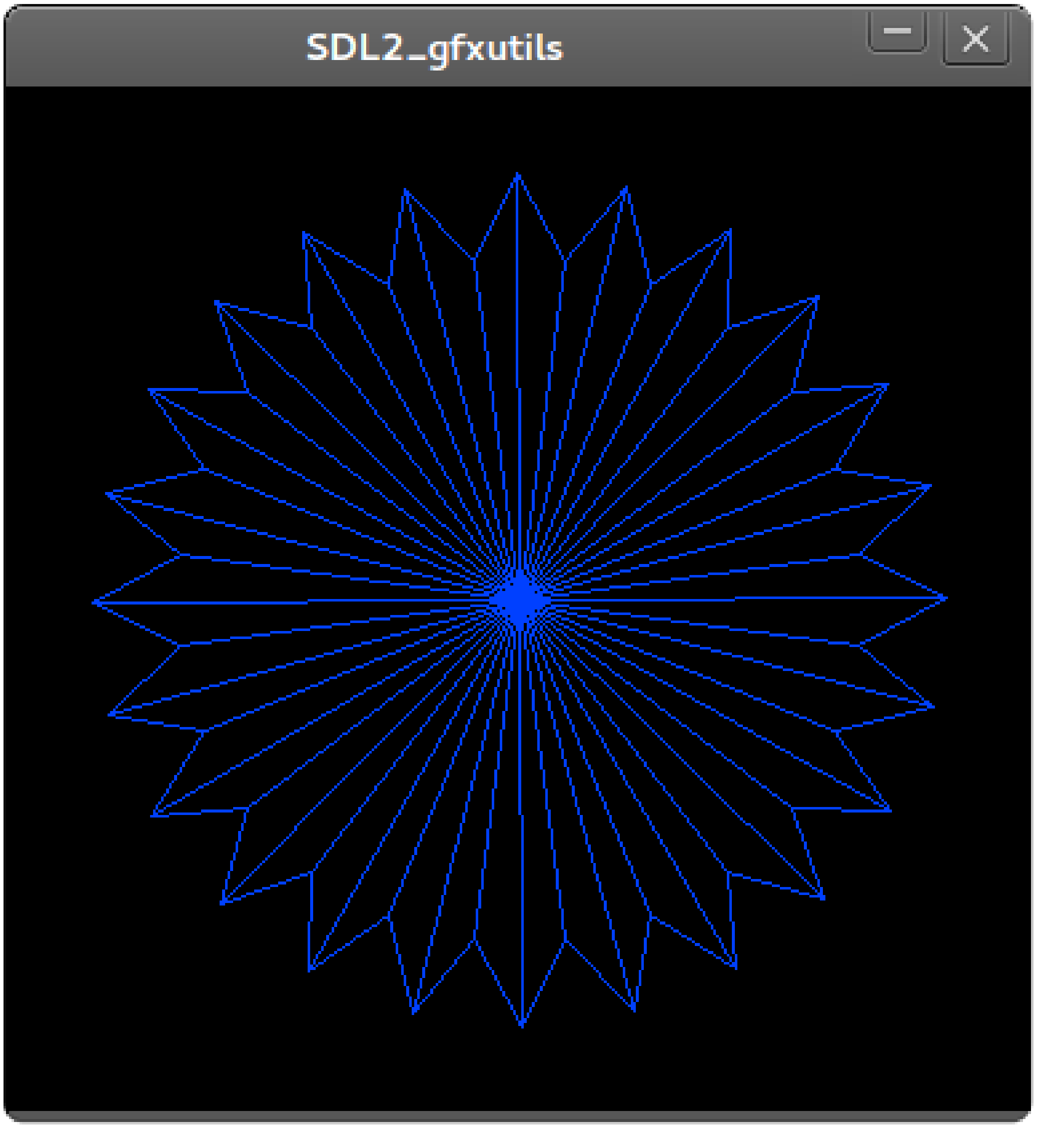


generate wheel

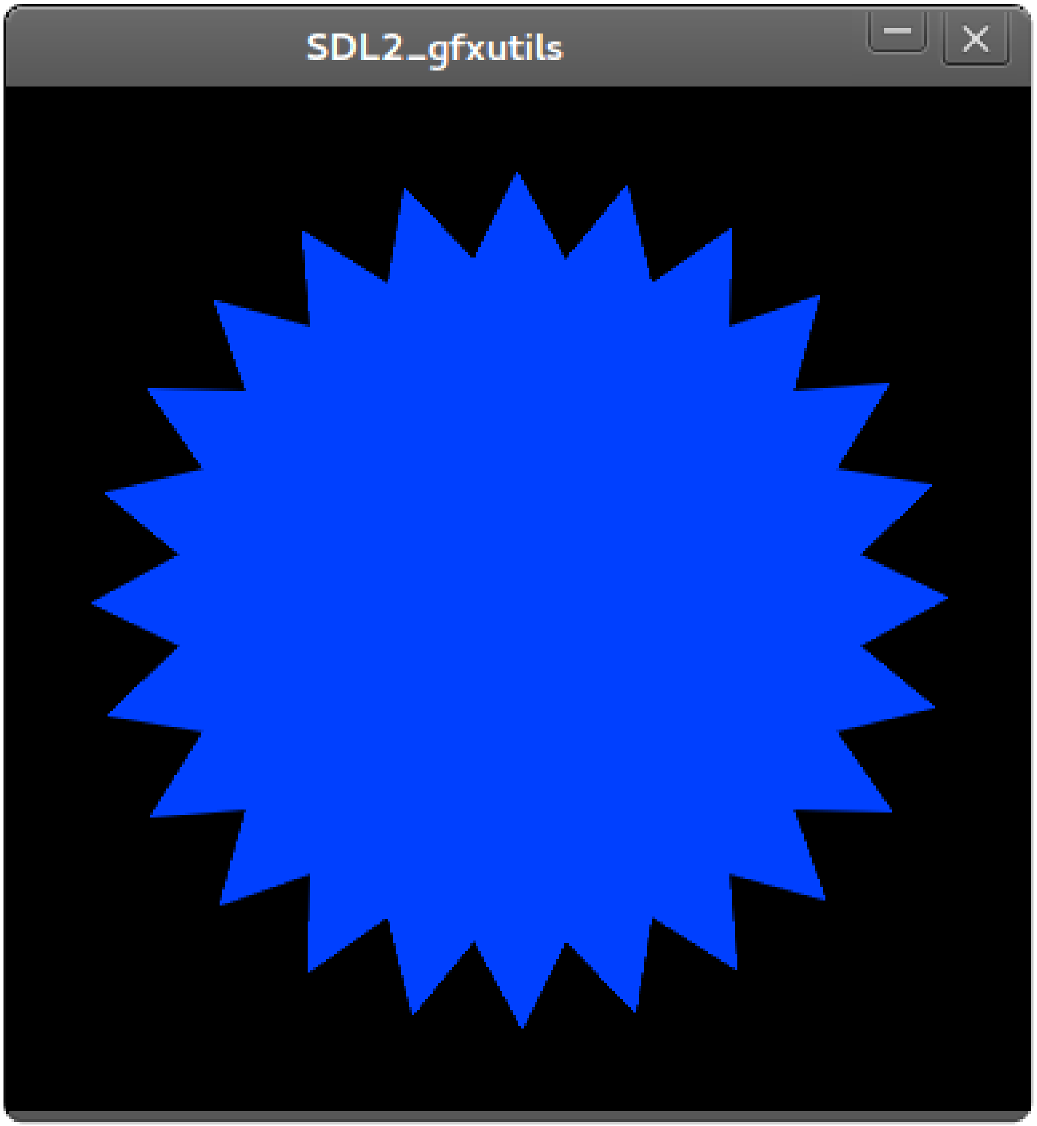
- display wheel



- display strikethrough wheel

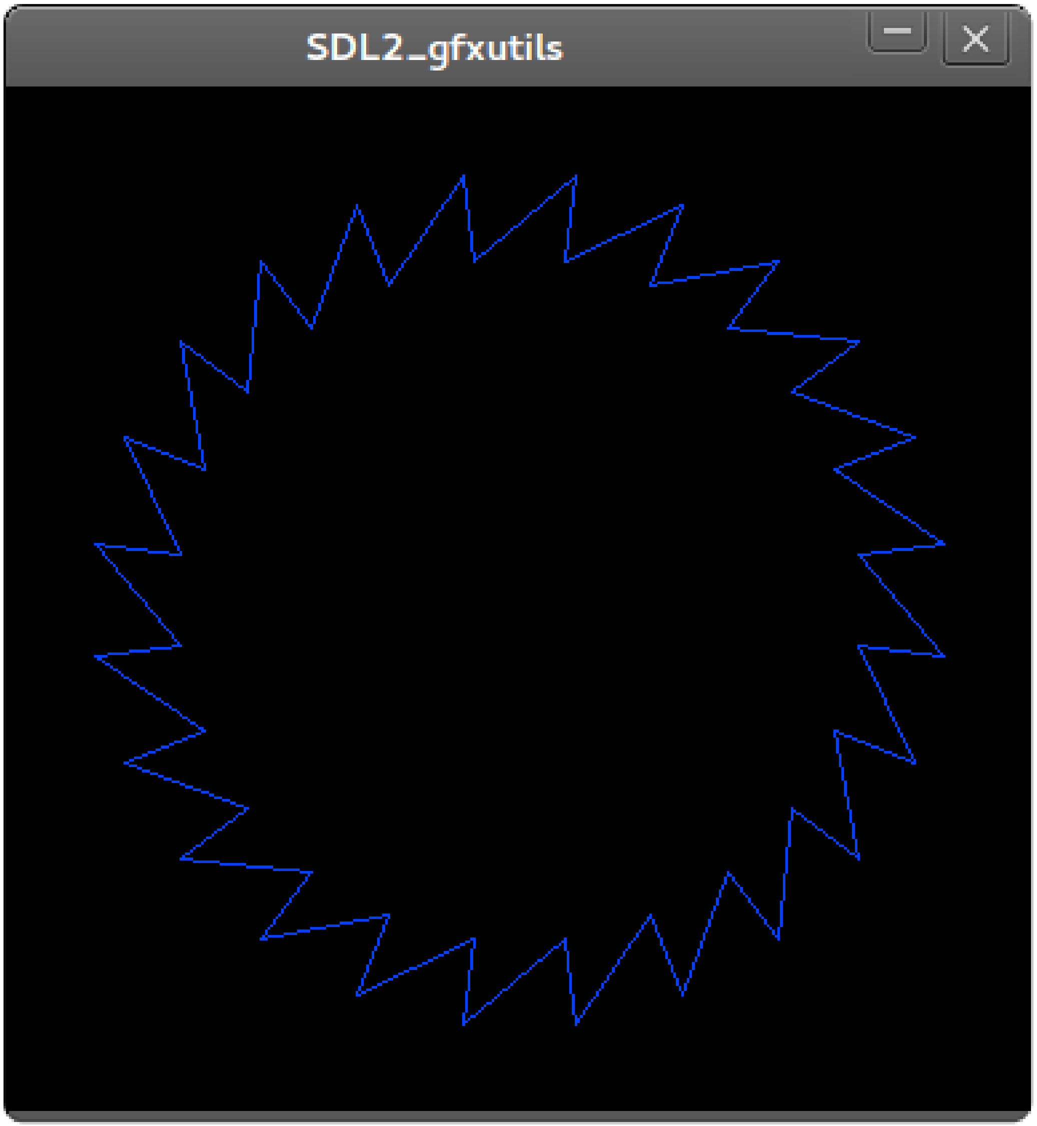


- display filled wheel

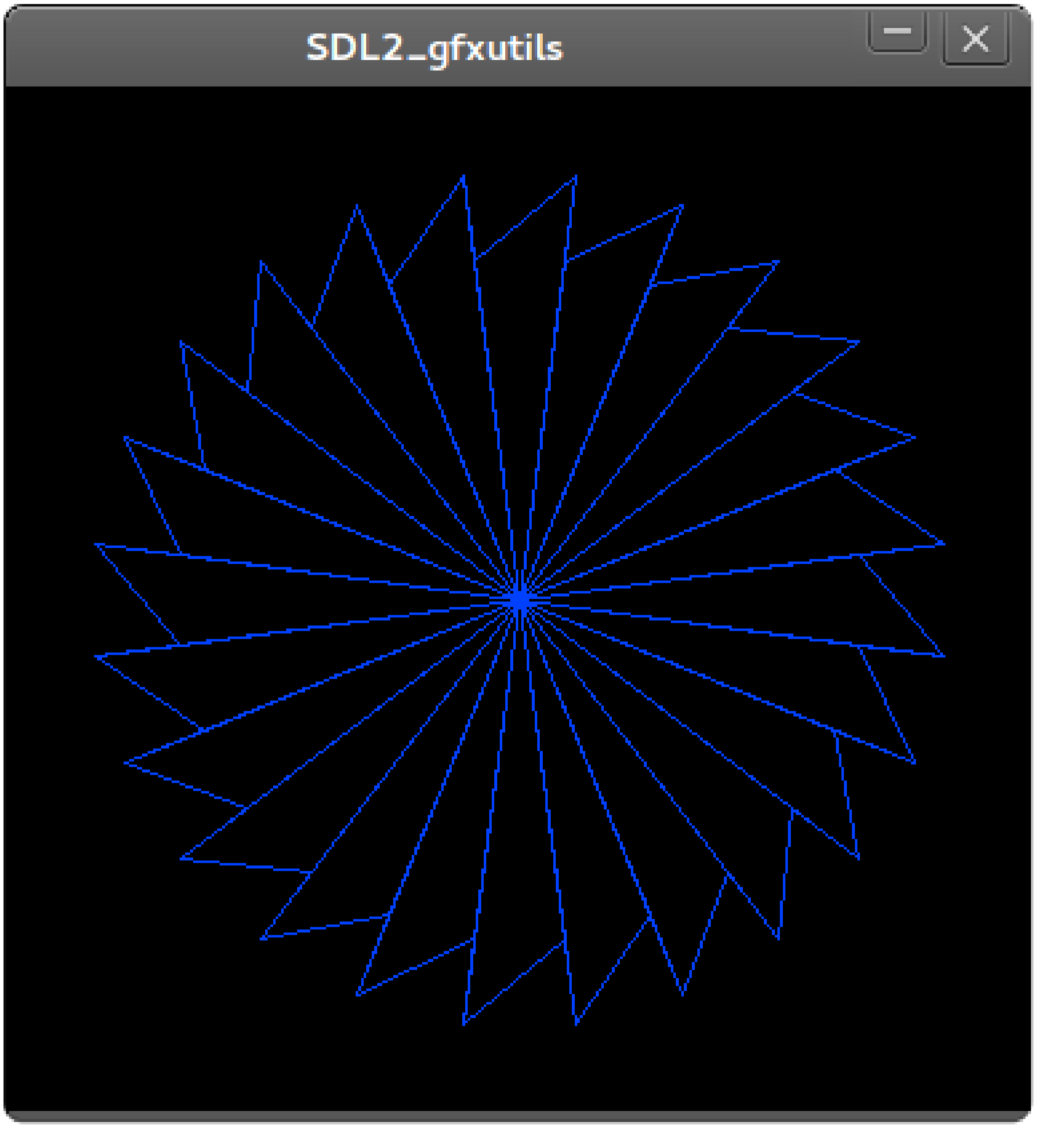


generate circular saw wheel

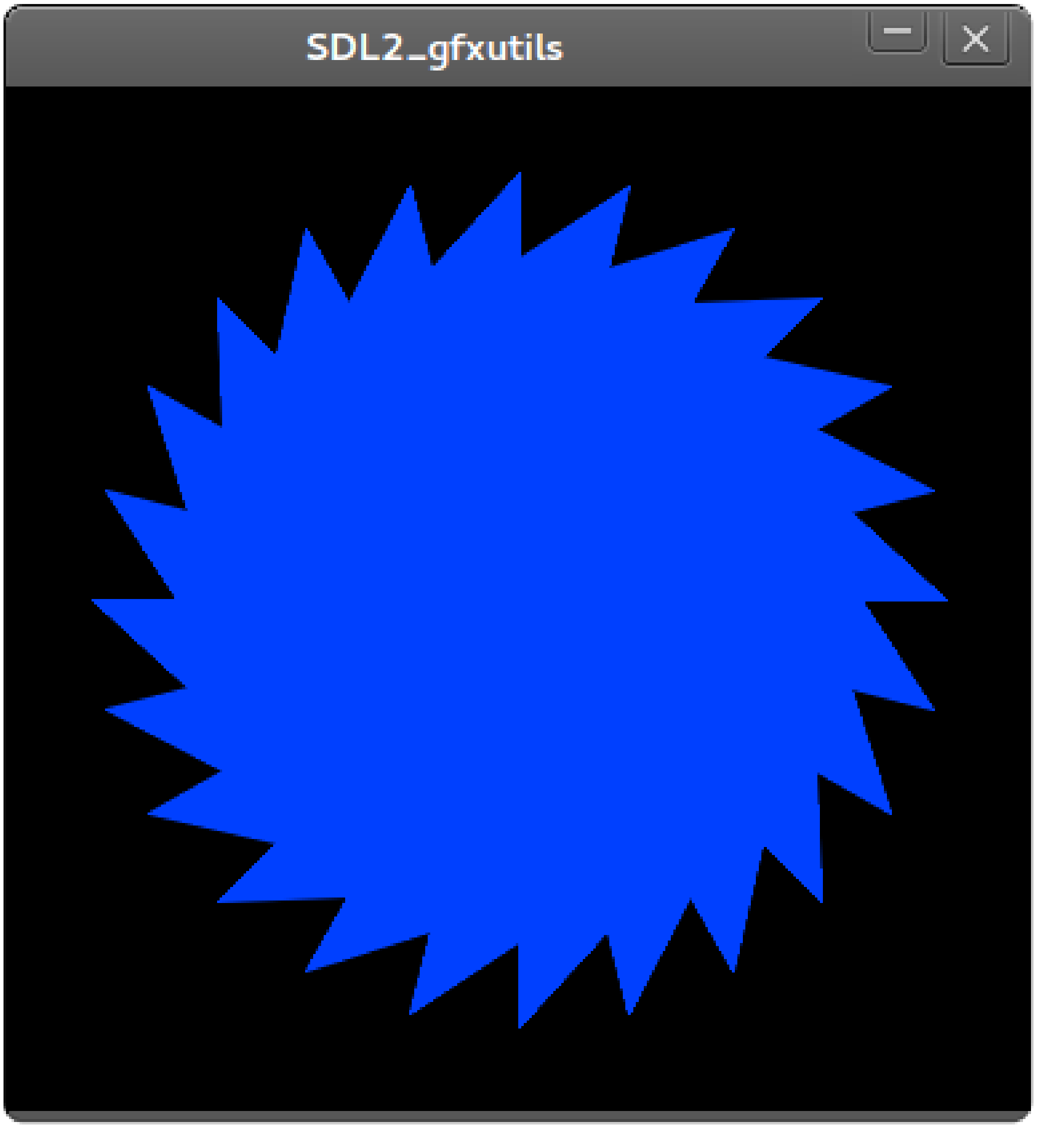
- display wheel



- display strikethrough wheel

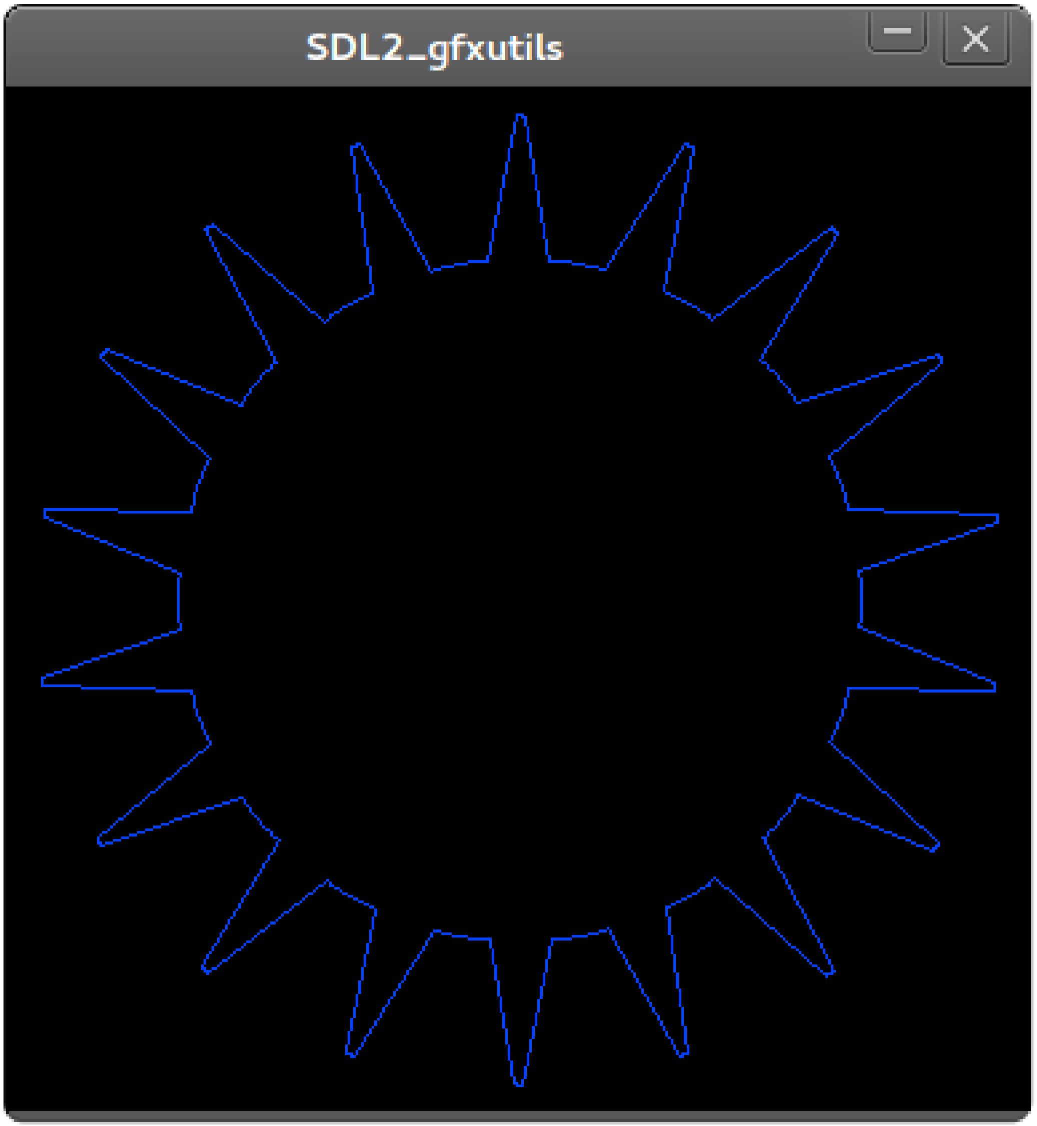


- display filled wheel

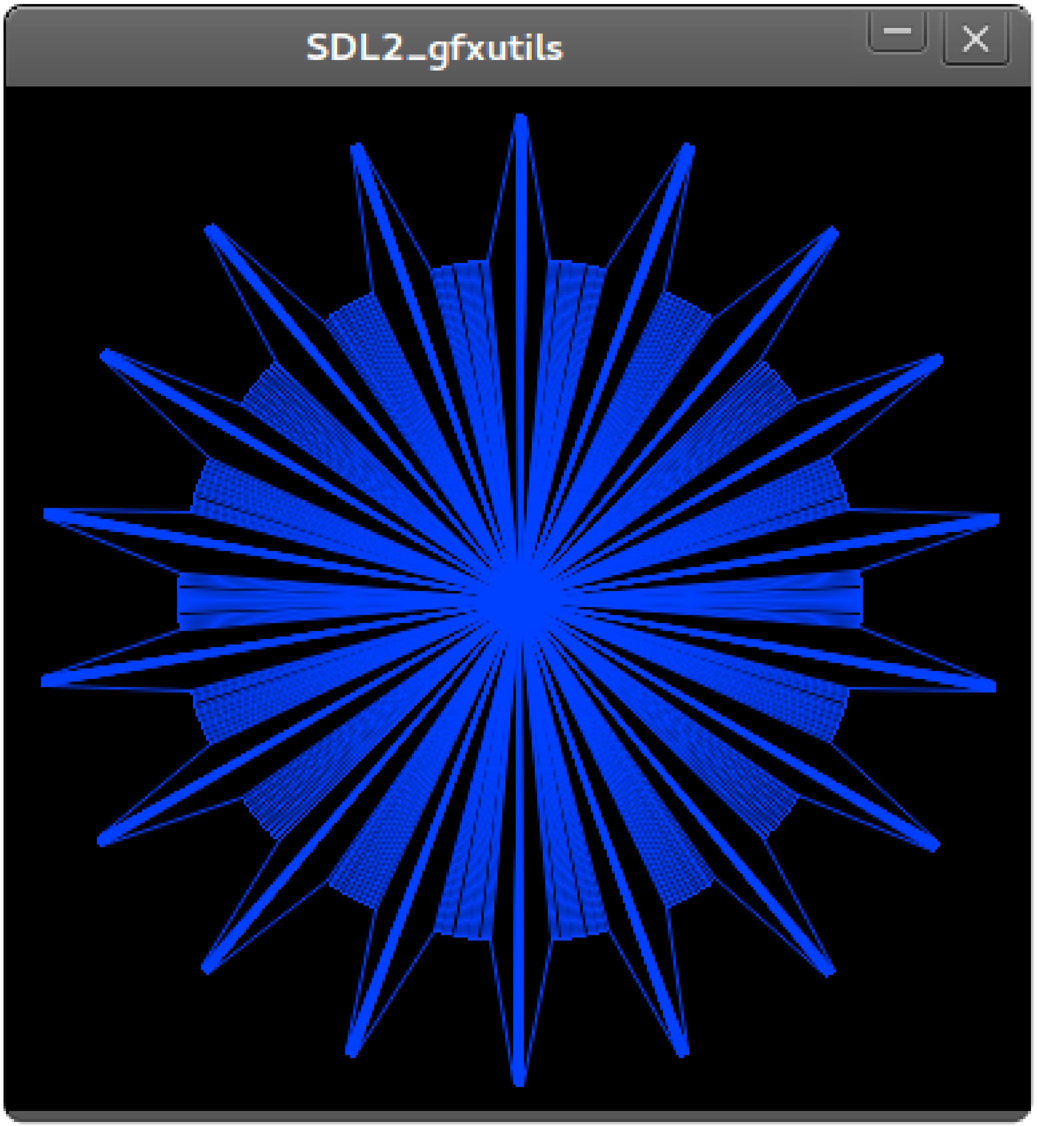


generate wheel peaks trigon

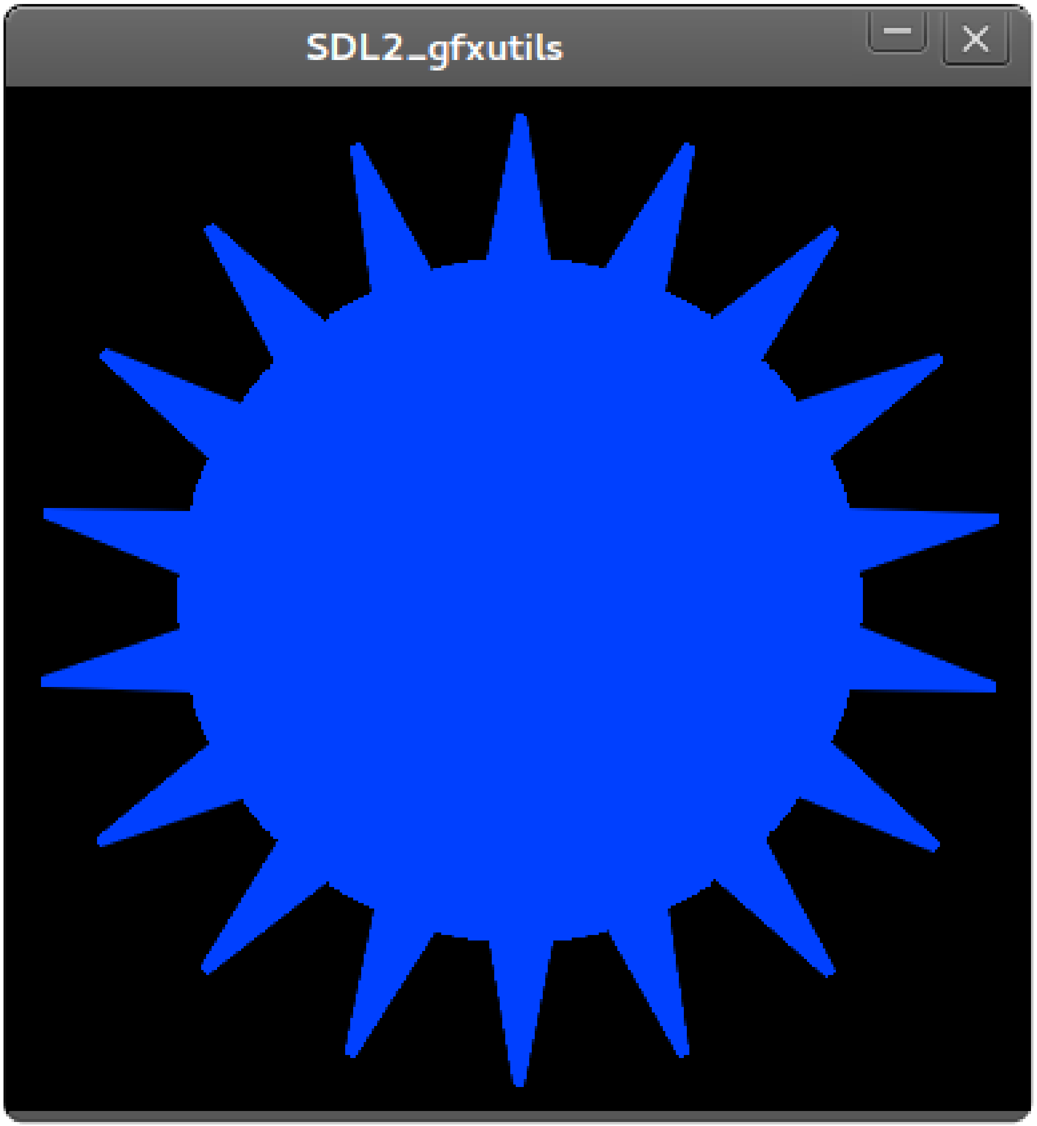
- display wheel



- display strikethrough wheel

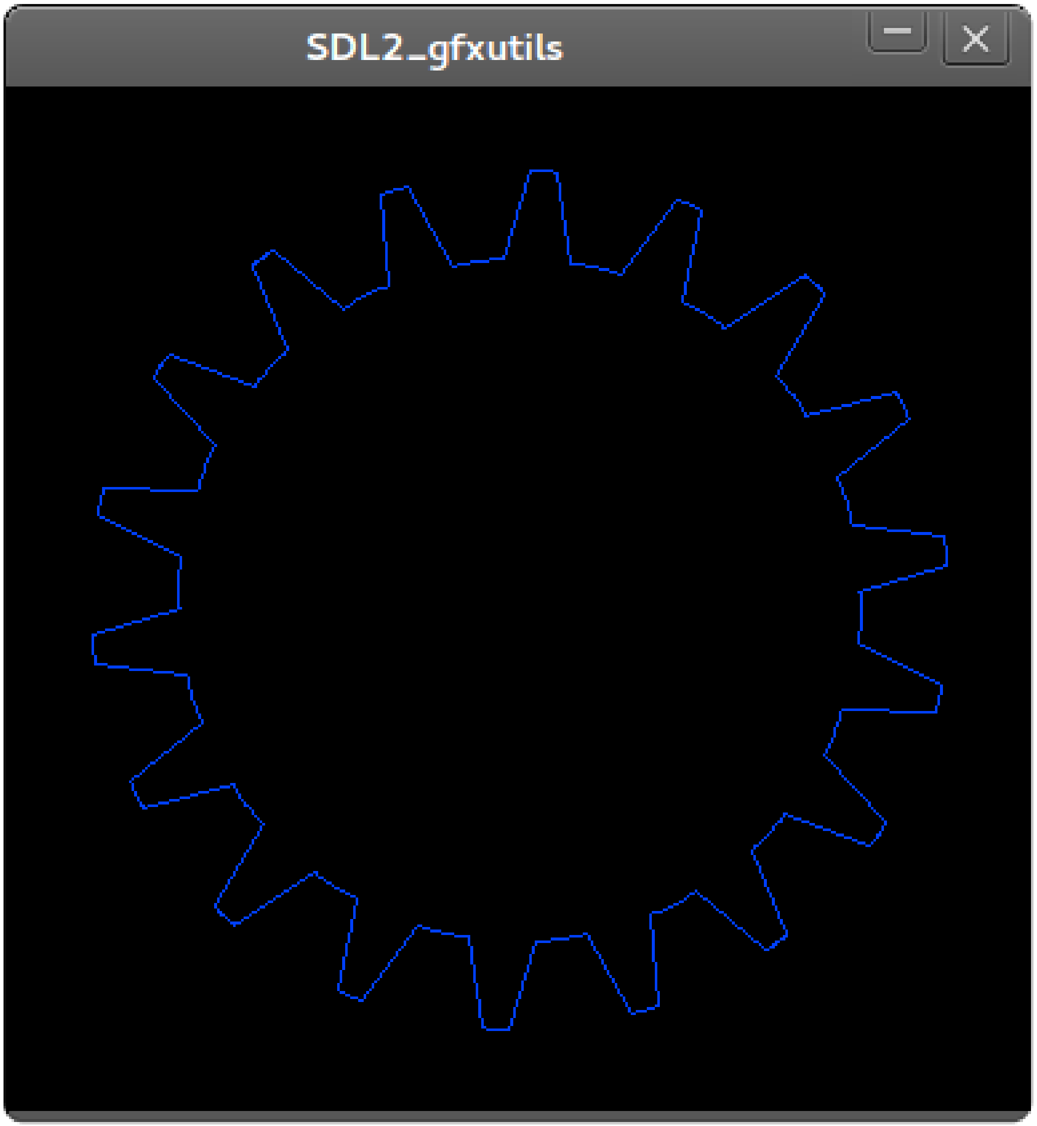


- display filled wheel

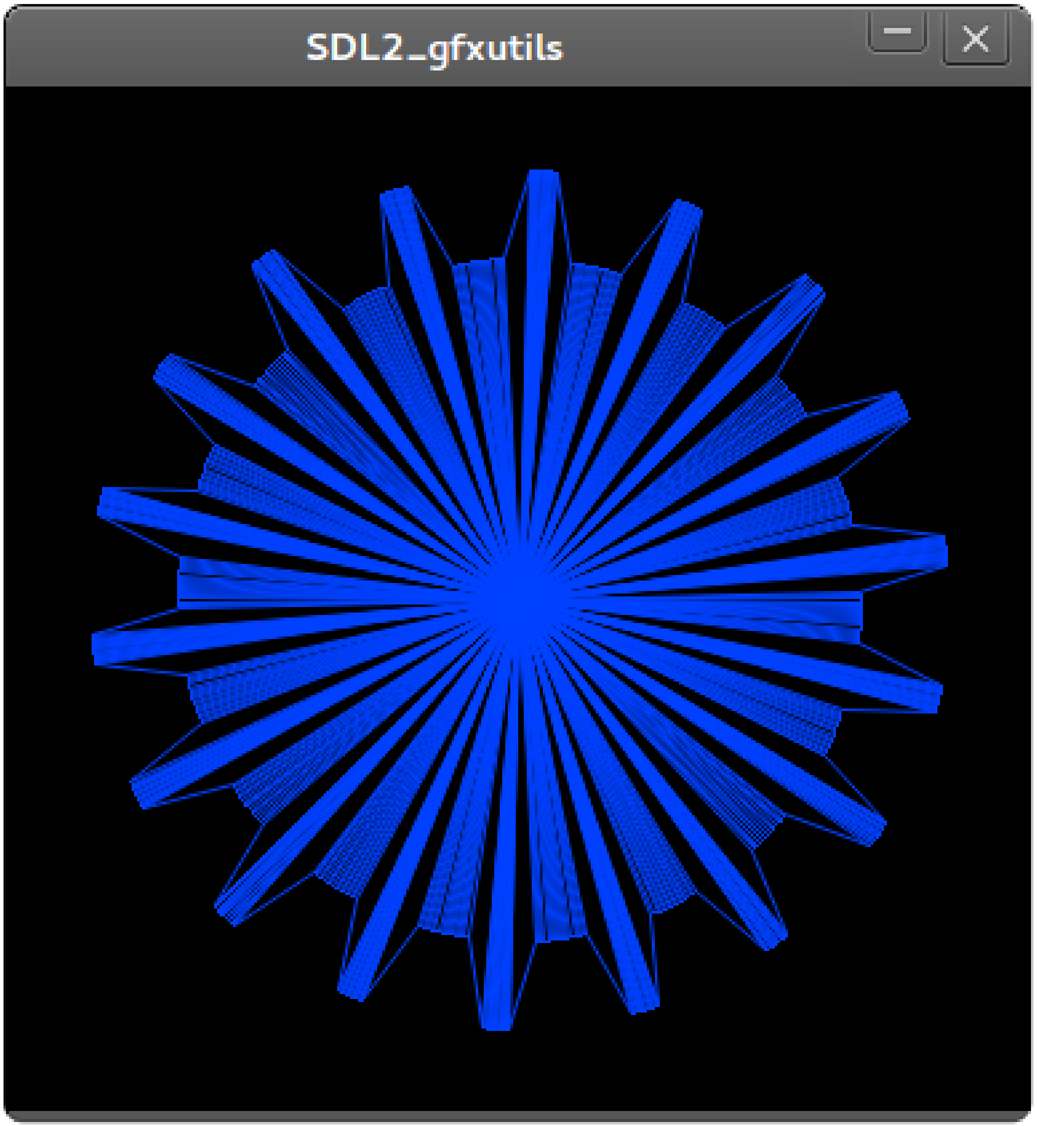


generate wheel peaks rounded square

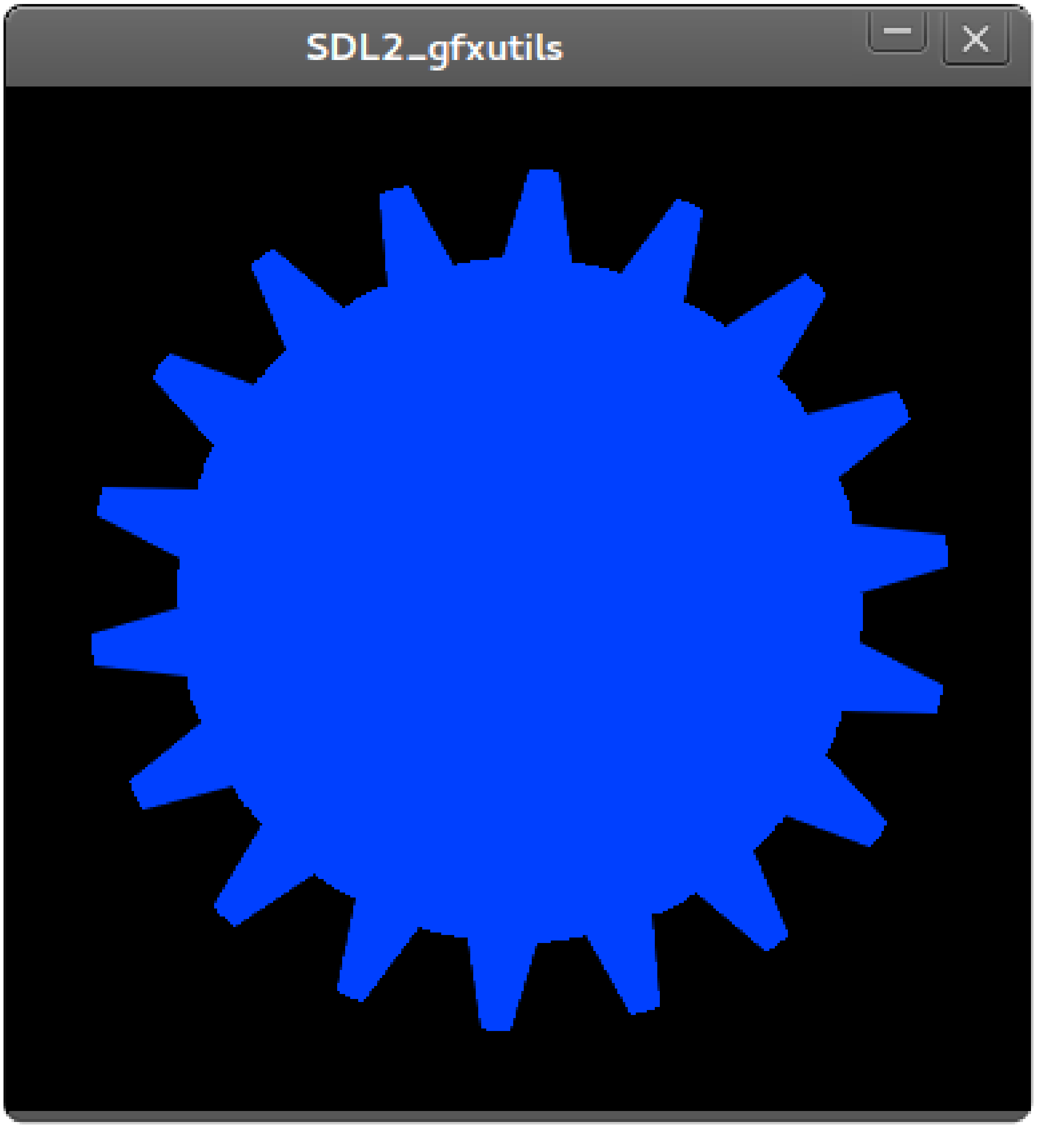
- display wheel



- display strikethrough wheel



- display filled wheel


SDL2_gfxutils

CHAPTER 12

SDL2_gfxutils header file

```
/************************************************************************************
 * SDL2_gfxutils a SDL2_gfx forms generating and manipulating helper functions set. *
 * Copyright (©) 2016 Brüggemann Eddie <mrcyberfighter@gmail.com>.                  *
 *                                                                                  *
 * This file is part of SDL2_gfxutils.                                              *
 * SDL2_gfxutils is free software: you can redistribute it and/or modify            *
 * it under the terms of the GNU General Public License as published by             *
 * the Free Software Foundation, either version 3 of the License, or                *
 * (at your option) any later version.                                              *
 *                                                                                  *
 * SDL2_gfxutils is distributed in the hope that it will be useful,                 *
 * but WITHOUT ANY WARRANTY; without even the implied warranty of                   *
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the                     *
 * GNU General Public License for more details.                                     *
 *                                                                                  *
 * You should have received a copy of the GNU General Public License                *
 * along with SDL2_gfxutils. If not, see <http://www.gnu.org/licenses/>             *
 ************************************************************************************/

#ifndef SDL2_GFXUTILS_HH  /** SDL2_gfxutils inclusion guard **/
#define SDL2_GFXUTILS_HH  /** SDL2_gfxutils inclusion guard **/

#include <SDL2/SDL.h>
#include <SDL2/SDL2_gfxPrimitives.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <stdint.h>
#include <math.h>

/* Définition of macro EXTERN_C for C++ compatibility */
#ifndef EXTERN_C
#ifdef __cplusplus
#define EXTERN_C extern "C"
#else
```

```
#define EXTERN_C
#endif
#endif

typedef struct Color_ {
  uint8_t r;
  uint8_t g;
  uint8_t b;
  uint8_t a;
} Color;


typedef struct Pixel_ {
  float x;
  float y;
} Pixel;

typedef struct Segment_ {
  Pixel xy1;
  Pixel xy2;
  Color color;
} Line;

typedef struct Coords_ {
  float *x;
  float *y;
} Coords;

typedef struct Polygon_ {
  Coords coords;
  Pixel center;
  Color color;
  uint16_t count;
  float length;
  float real_length;
  float orientation;
} Polygon;

typedef Polygon Arc;
typedef Polygon Hexagram;
typedef Polygon Pentagram;
typedef Polygon Star;
typedef Polygon Spiral;


typedef Polygon Form;
typedef Arc Form;
typedef Hexagram Form;
typedef Pentagram Form;
typedef Spiral Form;
typedef Star Form;



/** Base functions:
  ****************/
```

```
/** This function return an pixel initialized in relationship to the given settings.␣
↪**/
EXTERN_C Pixel get_pixel_coords(uint32_t position, uint32_t scale, float length,␣
↪Pixel center, float orientation);


/** This function compute the pixel middle point from the given line. **/
EXTERN_C Pixel get_middle_from_line(Line line);


/** Return a pointer on a Line starting at start_point, from length length, incline␣
↪from angle.
␣
↪********************************************************************************************************/
↪
EXTERN_C Line *generate_segment(Pixel start_point, float length, float angle);


/** Generate an arc from radius radius, from center center,
  * length from part of an circle circle_part,
  * starting at offset start_pos.
  ***************************************************************/
EXTERN_C Arc *generate_circle_arc(float radius, Pixel center, float start_pos, float␣
↪circle_part);

/**␣
↪**************************************************************************************************************␣
↪**/



/** Polygons:
  ***********/

/** Return an regular convex polygon according to the given settings.
  * with sides sides, with radius length radius, having for center center, incline␣
↪according orientation.
␣
↪**************************************************************************************************************
↪
EXTERN_C Polygon *generate_polygon_radius(uint32_t sides, float radius, Pixel center,␣
↪float orientation);

/** Generated an polygon which corners are arcs
  * with sides sides, with radius length radius, having for center center, incline␣
↪according orientation.
␣
↪**************************************************************************************************************
↪
EXTERN_C Polygon *generate_corners_rounded_polygon(uint32_t sides, float radius,␣
↪Pixel center, float orientation);

/** Generate an polygon which sides are arcs.
  * with sides sides, with radius length radius, having for center center, incline␣
↪according orientation.
␣
↪**************************************************************************************************************
↪
```

```
EXTERN_C Polygon *generate_sides_rounded_polygon(uint32_t sides, float radius, Pixel␣
↪center, float orientation);

/** Generated an rounded polygon alternating arcs rounded to the outside and to the␣
↪inside of the polygon.
  * with sides sides, with radius length radius, having for center center, incline␣
↪according orientation.
 ␣
↪**************************************************************************************************
↪
EXTERN_C Polygon *generate_rounded_inside_out_polygon(uint32_t sides, float radius,␣
↪Pixel center, float orientation);

/** Generated an polygon with half-circle rounded to the inside from the half sum␣
↪from the sides of the polygon
  * and the other half is even an arc or an straight line according to the side_arcs␣
↪boolean value.
  * with sides sides, with radius length radius, having for center center, incline␣
↪according orientation.
 ␣
↪**************************************************************************************************
↪
EXTERN_C Polygon *generate_alternate_inside_half_circle_polygon(uint32_t sides, float␣
↪radius, Pixel center, float orientation, bool side_arcs);

/** Generated an polygon with half-circle rounded to the outside from the half sum␣
↪from the sides of the polygon
  * and the other half is even an arc or an straight line according to the side_arcs␣
↪boolean value.
  * with sides sides, with radius length radius, having for center center, incline␣
↪according orientation.
 ␣
↪**************************************************************************************************
↪
EXTERN_C Polygon *generate_alternate_outside_half_circle_polygon(uint32_t sides,␣
↪float radius, Pixel center, float orientation, bool side_arcs);

/**␣
↪**************************************************************************************************
↪**/




/** Pentagram:
  ***********/

/** Generate an 5 extremity star with an centered pentagon from which every vertex go␣
↪to the center.
  * From radius radius, having for center center, incline according orientation.
 ␣
↪**************************************************************************************************
↪
EXTERN_C Pentagram *generate_pentagram(float radius, Pixel center, float orientation);

/** Generate an 5 extremity star.
  * With the particularity that the resulting star is not an regular star but an␣
↪pentagram star.
```

```
  * From radius radius, having for center center, incline according orientation.
 ␣
↪*********************************************************************************************/
↪
EXTERN_C Star *generate_pentagram_star(float radius, Pixel center, float orientation);

/** ********************************************************************** **/



/** Hexagram:
  ***********/

/** Generate an 5 extremity star with an centered hexagon from which every vertex go␣
↪to the center.
  * From radius radius, having for center center, incline according orientation.
 ␣
↪************************************************************************************************/
↪
EXTERN_C Hexagram *generate_hexagram(float radius, Pixel center, float orientation);

/** Generate an 6 extremity star.
  * With the particularity that the resulting star is not an regular star but an␣
↪hexagram star.
  * From radius radius, having for center center, incline according orientation.
 ␣
↪********************************************************************************************/
↪
EXTERN_C Star *generate_hexagram_star(float radius, Pixel center, float orientation);

/** ********************************************************************* **/



/** Stars:
  ********/

/** generate an simply star with the wanted settings:
 *  with pikes number of pikes,
 *  from radius radius,
 *  having for center center,
 *  incline according orientation.
 ****************************************************/
EXTERN_C Star *generate_star(uint32_t pikes, float radius, Pixel center, float␣
↪orientation);

/** ************************************************************************* **/



/** Wheels:
 *********/

/** Generate an pointed wheel accoridng the given settings.
  * With polygon as base polygon, having for center center, from radius radius,␣
↪incline according orientation.
 ␣
↪*****************************************************************************************************
↪
```

```
EXTERN_C Polygon *generate_wheel(uint32_t polygon, float radius, Pixel center, float␣
↪offset, float orientation);

/** Generate an circular saw like wheel.
  * With polygon as base polygon, having for center center, as points size offset␣
↪even reversed.
 ␣
↪**********************************************************************************************/
↪
EXTERN_C Polygon *generate_circular_saw_wheel(uint32_t polygon, float radius, Pixel␣
↪center, float offset, float orientation, bool reverse);

/** Generate an wheel (rounded polygon) with peaking as triangles which peaks ate␣
↪very little arcs.
  * With polygon as base polygon, having for center center, from peak size peak_
↪offset, incline according orientation.
 ␣
↪**************************************************************************************************
↪
EXTERN_C Polygon *generate_wheel_peaks_trigon(uint32_t sides, float radius, Pixel␣
↪center, float peak_offset, float orientation);

/** Generate an wheel (rounded polygon) with peaks looking like a tube but they are␣
↪only right-angled line to the sides connected trough an arc.
  * With polygon as base polygon, having for center center, from peak size peak_
↪length, incline according orientation.
 ␣
↪**************************************************************************************************
↪
EXTERN_C Polygon *generate_wheel_peaks_rounded_square(uint32_t sides, float radius,␣
↪Pixel center, float peak_length, float orientation);

/**␣
↪**************************************************************************************************
↪**/


/** Spiral:
  ********/

/** Generate a spiral.
  * making turns revolutions, having for center center, base rounded, with offset␣
↪betwen the turns offset_exponent.
 ␣
↪**************************************************************************************************
↪
EXTERN_C Spiral *generate_simple_spiral(Pixel center, uint32_t turns, uint32_t base,␣
↪float offset_exponent, float orientation, _Bool reverse);

/**␣
↪**************************************************************************************************
↪**/

/** fractal:
  *********/

/** Generate a star-like fractal.
  * With polygon as base polygon, having for center center, from radius radius,␣
↪incline according orientation, open change the issue form.
```

```
 ␣
↪**************************************************************************************************
↪
EXTERN_C Polygon *generate_fractal(uint32_t polygon, float radius, Pixel center,␣
↪float orientation, bool open);

/**␣
↪************************************************************************************************␣
↪**/



/** @Pixels operations:
  ********************/

/** Return a rotate a pixel around a center from the value angle in clock sens.
  *****************************************************************************/
EXTERN_C Pixel rotate(Pixel center, float angle, Pixel pixel);

/** Mirror a pixel on an axes.
  * pixel    = the pixel to mirror.
  * center   = the center for mirroring.
  * axes     = the mirror axes ['X'|'Y'].
  *****************************************/
EXTERN_C Pixel mirror(Pixel pixel, Pixel center, char axes);

/** Return the new position from pixel scaled by factor:
  * factor < 1 == scaling littler.
  * factor > 1 == scaling greater.
  ******************************************************/
EXTERN_C Pixel scale(Pixel center, float factor, Pixel pixel);

/** Return a translated pixel from value x and y.
  ***********************************************/
EXTERN_C Pixel translate(Pixel pixel, float x, float y);

/** *********************************************** **/



/** @Forms operations:
  *******************/

/** Rotate a form from angles degrees.
  ************************************/
EXTERN_C void rotate_form(Form * form, float angle);

/** Mirror a Form through the axes axes ['X'|'Y'].
  ************************************************/
EXTERN_C void mirror_form(Form * form, Pixel center, char axes);

/** Scale a Form from factor factor.
  * if factor > 1.0 the size of the form increase.
  * if factor < 1.0 the size from the form decrease.
  **************************************************/
EXTERN_C void scale_form(Form * form, float factor);

/** Translate a Form from (x, y) pixels .
```

```
  ****************************************/
EXTERN_C void translate_form(Form * form, float x, float y);

/** Remove doubles coordinates from Form from.
  ********************************************/
EXTERN_C Form *remove_doubles_form(Form * form);

/** ************************************************ **/



/** Setters:
  **********/

/** Set a new center to from Form form and
  * even translate all the form according to the new center.
  **********************************************************/
EXTERN_C void set_form_center(Form * form, Pixel center, bool translate);

/** Set a new radius to from Form form and
  * scale the form according to the new radius.
  *********************************************/
EXTERN_C void set_form_radius(Form * form, float radius);

/** Set the colors of the Form form.
  **********************************/
EXTERN_C void set_form_color(Form * form, uint8_t red, uint8_t green, uint8_t blue,␣
↪uint8_t alpha);

/** Set the colors of the Line line.
  **********************************/
EXTERN_C void set_line_color(Line * line, uint8_t red, uint8_t green, uint8_t blue,␣
↪uint8_t alpha);

/**␣
↪***********************************************************************************␣
↪**/



/** Getters:
  *********/

/** Return the current center from the Form form.
  ***********************************************/
EXTERN_C Pixel get_form_center(Form * form);

/** Return the current color from the Form form.
  ***********************************************/
EXTERN_C Color get_form_color(Form * form);

/** Return the current length from the Form form.
  * The length member is often the radius @see documentation.
  ***********************************************************/
EXTERN_C float get_form_length(Form * form);

/** Return the current real length from the Form form.
  * The real length member is the distance between the center and the farest point␣
↪from it.
```

```
 ␣
↪*****************************************************************************************/
↪
EXTERN_C float get_form_real_length(Form * form);

/** Return the current orientation from the Form form.
  ****************************************************/
EXTERN_C float get_form_orientation(Form * form);

/** ********************************* **/



/** Geometry utils:
  ****************/

/** Return the angle for the given arguments.
  *******************************************/
EXTERN_C float get_angle(int position, float scale, float orientation);

/** Return the distance between px1(x,y) and px2(x,y).
  ****************************************************/
EXTERN_C float get_distance_pixels(Pixel px1, Pixel px2);

/** ******************************************************** **/



/** Displaying forms:
 *******************/

/** @Forms normal displaying:
  **************************/

/** Display the Line line according to his settings
  * @return 0 on success, -1 on failure.
  *************************************************/
EXTERN_C int display_line(SDL_Renderer * pRenderer, Line * line);

/** Display the Arc arc according to his settings
  * @return 0 on success, -1 on failure.
  ************************************************/
EXTERN_C int display_arc(SDL_Renderer * pRenderer, Arc * arc);

/** Display the Form polygon according to his settings
  * @return 0 on success, -1 on failure.
  ****************************************************/
EXTERN_C int display_polygon(SDL_Renderer * pRenderer, Form * polygon);

/** Display the Form polygon strikethrough according to his settings
  * @return 0 on success, -1 on failure.
  ******************************************************************/
EXTERN_C int display_strikethrough_polygon(SDL_Renderer * pRenderer, Form * polygon);

/** Display the Form polygon filled according to his settings
  * @return 0 on success, -1 on failure.
  ***********************************************************/
EXTERN_C int display_filled_polygon(SDL_Renderer * pRenderer, Form * polygon);
```

```
/** Display the Pentagram pentagram according to his settings
  * @return 0 on success, -1 on failure.
  ***********************************************************/
EXTERN_C int display_pentagram(SDL_Renderer * pRenderer, Pentagram * pentagram);

/** Display the Hexagram Hexagram according to his settings
  * @return 0 on success, -1 on failure.
  *********************************************************/
EXTERN_C int display_hexagram(SDL_Renderer * pRenderer, Hexagram * hexagram);

/** Display the Star star according to his settings
  * @return 0 on success, -1 on failure.
  *************************************************/
EXTERN_C int display_star(SDL_Renderer * pRenderer, Star * star);

/** Display the Star star flower-like according to his settings
  * @return 0 on success, -1 on failure.
  **************************************************************/
EXTERN_C int display_flower_star(SDL_Renderer * pRenderer, Star * star);

/** Display the Star star strikethrough according to his settings
  * @return 0 on success, -1 on failure.
  ***************************************************************/
EXTERN_C int display_strikethrough_star(SDL_Renderer * pRenderer, Star * star_
↪striketrough);

/** Display the Star star polygon according to his settings
  * @return 0 on success, -1 on failure.
  *********************************************************/
EXTERN_C int display_polygon_star(SDL_Renderer * pRenderer, Star * star);

/** Display the Star star filled according to his settings
  * @return 0 on success, -1 on failure.
  ********************************************************/
EXTERN_C int display_filled_star(SDL_Renderer * pRenderer, Star * star);

/** Display the Spiral spiral according to his settings
  * @return 0 on success, -1 on failure.
  *****************************************************/
EXTERN_C int display_spiral(SDL_Renderer * pRenderer, Spiral * spiral);

/** *************************************************************************** **/



/** @Forms thickness displaying:
  ******************************/

/** Display the Line line according to his settings
  * @return 0 on success, -1 on failure.
  *************************************************/
EXTERN_C int display_line_thick(SDL_Renderer * pRenderer, Line * line, uint8_t␣
↪thickness);

/** Display the Arc arc according to his settings
  * @return 0 on success, -1 on failure.
  ************************************************/
EXTERN_C int display_arc_thick(SDL_Renderer * pRenderer, Arc * arc, uint8_t␣
↪thickness);

/** Display the Form polygon according to his settings
  * @return 0 on success, -1 on failure.
  ****************************************************/
EXTERN_C int display_polygon_thick(SDL_Renderer * pRenderer, Form * polygon, uint8_t␣
↪thickness);

/** Display the Form polygon strikethrough according to his settings
  * @return 0 on success, -1 on failure.
  ******************************************************************/
EXTERN_C int display_strikethrough_polygon_thick(SDL_Renderer * pRenderer, Form *␣
↪polygon, uint8_t thickness);

/** Display the Pentagram pentagram according to his settings
  * @return 0 on success, -1 on failure.
  ***********************************************************/
EXTERN_C int display_pentagram_thick(SDL_Renderer * pRenderer, Pentagram * pentagram,␣
↪uint8_t thickness);

/** Display the Hexagram Hexagram according to his settings
  * @return 0 on success, -1 on failure.
  *********************************************************/
EXTERN_C int display_hexagram_thick(SDL_Renderer * pRenderer, Hexagram * hexagram,␣
↪uint8_t thickness);

/** Display the Star star according to his settings
  * @return 0 on success, -1 on failure.
  *************************************************/
EXTERN_C int display_star_thick(SDL_Renderer * pRenderer, Star * star, uint8_t␣
↪thickness);

/** Display the Star star flower-like according to his settings
  * @return 0 on success, -1 on failure.
  **************************************************************/
EXTERN_C int display_flower_star_thick(SDL_Renderer * pRenderer, Star * star, uint8_t␣
↪thickness);

/** Display the Star star strikethrough according to his settings
  * @return 0 on success, -1 on failure.
  ***************************************************************/
EXTERN_C int display_strikethrough_star_thick(SDL_Renderer * pRenderer, Star * star_
↪striketrough, uint8_t thickness);

/** Display the Star star polygon according to his settings
  * @return 0 on success, -1 on failure.
  *********************************************************/
EXTERN_C int display_polygon_star_thick(SDL_Renderer * pRenderer, Star * star, uint8_
↪t thickness);

/** Display the Spiral spiral according to his settings
  * @return 0 on success, -1 on failure.
  *****************************************************/
EXTERN_C int display_spiral_thick(SDL_Renderer * pRenderer, Spiral * spiral, uint8_t␣
↪thickness);
```

```
/**␣
↪****************************************************************************************************
↪**/



/** @Forms anti-aliasing displaying:
  *********************************/

/** Display the Line line according to his settings
  * @return 0 on success, -1 on failure.
  **************************************************/
EXTERN_C int aa_display_line(SDL_Renderer * pRenderer, Line * line);

/** Display the Arc arc according to his settings
  * @return 0 on success, -1 on failure.
  *************************************************/
EXTERN_C int aa_display_arc(SDL_Renderer * pRenderer, Arc * arc);

/** Display the Form polygon according to his settings
  * @return 0 on success, -1 on failure.
  *****************************************************/
EXTERN_C int aa_display_polygon(SDL_Renderer * pRenderer, Form * polygon);

/** Display the Form polygon strikethrough according to his settings
  * @return 0 on success, -1 on failure.
  *******************************************************************/
EXTERN_C int aa_display_strikethrough_polygon(SDL_Renderer * pRenderer, Form *␣
↪polygon);

/** Display the Pentagram pentagram according to his settings
  * @return 0 on success, -1 on failure.
  ************************************************************/
EXTERN_C int aa_display_pentagram(SDL_Renderer * pRenderer, Pentagram * pentagram);

/** Display the Hexagram Hexagram according to his settings
  * @return 0 on success, -1 on failure.
  **********************************************************/
EXTERN_C int aa_display_hexagram(SDL_Renderer * pRenderer, Pentagram * hexagram);

/** Display the Star star according to his settings
  * @return 0 on success, -1 on failure.
  **************************************************/
EXTERN_C int aa_display_star(SDL_Renderer * pRenderer, Star * star);

/** Display the Star star flower-like according to his settings
  * @return 0 on success, -1 on failure.
  ***************************************************************/
EXTERN_C int aa_display_flower_star(SDL_Renderer * pRenderer, Star * star);

/** Display the Star star strikethrough according to his settings
  * @return 0 on success, -1 on failure.
  ****************************************************************/
EXTERN_C int aa_display_strikethrough_star(SDL_Renderer * pRenderer, Star * star);

/** Display the Star star polygon according to his settings
  * @return 0 on success, -1 on failure.
  **********************************************************/
```

```
EXTERN_C int aa_display_polygon_star(SDL_Renderer * pRenderer, Star * star);

/** Display the Spiral spiral according to his settings
  * @return 0 on success, -1 on failure.
  *****************************************************/
EXTERN_C int aa_display_spiral(SDL_Renderer * pRenderer, Spiral * spiral);

/** *********************************************************************** **/



/** Memory:
  ********/

/** Free the given Form form
  * 1. the coordinates arrays.
  * 2. the form.
  * And set the pointer on NULL
  *****************************/
EXTERN_C void free_form(Form * form);

/** Allocate space for a new Form:
  *******************************/
EXTERN_C Form *new_form(uint32_t count);

/** ************************* **/



/** Utils:
  *******/

/** Check if the SDL2_Renderer is valid.
  **************************************/
EXTERN_C void check_renderer(SDL_Renderer * pRenderer);

/** Check if form != NULL
  ***********************/
EXTERN_C void check_form(Form * form);

/** *************************************** **/


/** Miscealeanous:
  ***************/

/** Generate an animation guideline.
  * By filling an Pixel array.
  **********************************/
EXTERN_C void compute_trajectory(Pixel positions[], Line * trajectory, uint32_t␣
↪steps);

#endif  /** SDL2_gfxutils inclusion guard **/
```

CHAPTER 13

Indices and tables

- genindex
- modindex
- search

Index