**Background:**

Geometry in the Cartesian plane is generally well-understood. We’re comfortable working with various shapes and can deduce many things about area, slope, and angle measure. Generally speaking though, we don’t have very much to say about the real line (which forms the basis for this geometry) other than simple observations. What if we formed a similar geometry using a more well-understood and tangible set, such as the integers? This question gives rise to the **Lattice Plane** (), which is the set of ordered pairs where each coordinate is an integer.

At first glance, we might expect many properties to carry over from to After all, the lattice plane is simply a subset of the familiar Cartesian plane. Indeed, many properties carry over, but the restriction to the integer points allows us to discover specialized properties, unique to the lattice. Today, we will explore the connection between **primitive lattice triangles** and **primitive lattice parallelograms.**

In lattice geometry, a polygon is known as **primitive** if the polygon contains no lattice points other than its vertices, including on the sides and interior of the polygon. Today we will create a helpful visual that can be used in constructing a primitive lattice parallelogram given a primitive lattice triangle.

|  |  |
| --- | --- |
| General Goal: We will create the parallelogram by rotating the triangle in the correct way to give us the full parallelogram. We need to make sure the lattice is preserved in this process (i.e. lattice points remain lattice points and non-lattice points remain non-lattice points in the rotation). |  |
| To begin, we need a way to describe the beginning triangle and the rotated triangle using sets. Recall that the set description of a parallelogram is where *a* and *b* are real numbers and ***v*** and ***w*** are the vectors emanating from the origin which form the sides of the original triangle. |  |
| Given that set description, how can we describe the lower and upper triangles using similar sets? It turns out that the lower triangle is and the upper triangle is |  |
| This is sort of random though. How can we convince ourselves that the sets we came up with are truly the correct sets? We will create a GeoGebra project which will trace out all possible points within both sets to see the sets are correct. Begin by creating an arbitrary (doesn’t need to be primitive) lattice triangle. Make sure the vertices are all lattice points! Then create the fourth vertex to form the parallelogram by adding the two vectors which form the triangle. TIP: to access the coordinates of a point in GeoGebra, use x(A) or y(A) to get the x or y coordinates of point A respectively. |  |
| We will first create our “brush” for the lower triangle. We will use a point which is shown only if it meets the restrictions for a point to be in the lower triangle which we described above. Then, we will animate the point and trace the point as it fills the lower triangle. First, create two sliders *a* and *b* which have minimum value 0 and maximum value 1. These will be the coefficients of our vectors *v* and *w* within the set . Also set the increment for each to 0.001. This will allow us to see more tracings once we animate the points. |  |
| Now we will create an element of the set . Create a point To show this point only if it meets the criteria to be a member of , click on the options for point , go to the “Advanced” tab, and click the field “Condition to Show Object”. Recall that in order for to be in we need our coefficients *a* and *b* to satisfy . Simply type this into the field. Try moving the sliders around to see the sorts of values that lie in the bottom triangle. |  |
| Now we will trace out the lower triangle using point *D*. In the “Basic” tab on the option menu for *D*, click “Show Trace”. Now go to the options menu for slider *b* and under the “Slider” tab, and change the speed to 10. Create a button called “Lower Triangle” and use the commands “StartAnimation(a)” and “StartAnimation(b)”. If you would like to stop the animations as well, create a button called “Stop Lower Animation” and use the commands “StartAnimation(a, 1>2)” and “StartAnimation(b, 1>2)”. |  |
| Make sure the options menu is closed. Now, if you click the button “Lower Triangle”, you will see point *D* begin to trace out the lower triangle. To make it more colorful, change the color of point *D*. What have we just done? Every traced point which is now visible satisfies the condition to be in our set If you let the animation go on long enough, you will see that the trace fills all of We should feel much more comfortable now claiming that the lower triangle is, as a set, equal to |  |
| Repeat the steps above to create a new point *E* that traces out the upper triangle. The process is the same except for the condition to show point *E*. Try to make point *E* and figure out the condition to show *E*! |  |

Now that we have both a visualization and a set description of our two triangles, we can proceed with the rotation itself. The above steps might seem unrelated to the rotation, but having a rigorous set definition of the two triangles is essential in the proof of our construction, specifically in the proof that the lattice is preserved in the rotation we’re about to do.

|  |  |
| --- | --- |
| Begin by constructing a triangle like the one you were working with in the previous steps. Note that it does not have to be primitive. |  |
| We will rotate the entire triangle about the midpoint of the segment between points A and B. Use the construct midpoint tool to construct midpoint D. |  |
| We want to animate the rotation to see the points moving throughout the map. Create a slider called α with an angle value, which ranges between 0° and 180°, and under the animation tab change oscillating to increasing only once. |  |
| Now under the transform tab, hit rotate around point. Then select your triangle and point D (in that order). In the popup menu, delete the default angle of 45° and use the Greek keyboard to enter α. Try moving slider α to see various stages of the rotation. |  |
| Now let’s animate α! Create a new button with the command “StartAnimation(α)”. If you’d like to reset the picture, create a reset button as well with the command “α=0”. To see that the rotated triangle ends up where we want, show the value of the upper right vertex and check that it ends up at the sum of the two vectors from the lower left triangle. |  |
| Great! We’ve created a parallelogram from a triangle. How can we convince ourselves that if we start with a primitive lattice triangle, we end up with a primitive lattice parallelogram? Start by making two points within the lower triangle: one lattice, one non-lattice. Make them distinct colors and show their values. |  |
| Now rotate both points around D just as you rotated the triangle. What happens when α = 180°? The image of the lattice point is a lattice point and the image of a non-lattice point is a non-lattice point! Wow! Try moving each point to different points within the lower triangle. |  |
| Ok, but how does this help us in the primitive case? Remember, we want to show that the rotated version of the triangle is still primitive, and thus the parallelogram is also primitive. Well, if we begin with a primitive triangle, we just showed under the rotation that any lattice points in the triangle (just the vertices here) *must* go to lattice points in the rotated triangle. Also, any non-lattice points must *remain* non-lattice points after the rotation. So if the starter triangle was primitive, the new triangle must also be! How great! |  |

Note that what we’ve done today does not constitute a rigorous proof. We’ve simply provided a nice visualization that might help others understand an eventual proof. If you want to try out the actual proof, you’ll need to break the rotation into three separate maps: first a translation of the plane so that the midpoint becomes the origin, then a linear map to rotate the plane, then a translation back to the correct place. If you’d like to see a formal proof of this construction, see the exercise (rotate.pdf) on my github:

<https://github.com/collinschad/Math300>

The problem presented there is slightly different from the problem we talked about today, but the argument contains the proof of the construction.