Visualización en la Criptografía basada en Retículas.

Jaziel D. Flores Rodríguez[1], Miguel E. Pérez Ibarra[1], Fernando Q. Valencia Rodríguez[2], Alfonso F. De Abiega L'Eglisse[3], and Gina Gallegos-García[4]

[1] Escuela Superior de Física y Matemáticas, Av. Instituto Politécnico Nacional Edificio 9, Unidad Profesional Adolfo López Mateos, Zacatenco, Delegación Gustavo A. Madero, C.P. 07738, CDMX

[2] Escuela Superior de Cómputo, Av. Juan de Dios Bátiz s/n esq. Av. Miguel Othón de Mendizabal. Colonia Lindavista. Delegación Gustavo A. Madero. C.P. 07738. CDMX

[3] Escuela Superior de Ingeniería Mécanica y Electrónica Unidad Culhuacan, Avenida Santa Ana 1000, San Francisco Culhuacan, Culhuacan CTM V, Delegación Coyoacán, 04440,CDMX

[4] Centro de Investigación en Computación, Av. Juan de Dios Bátiz S/N, Nueva Industrial Vallejo, Delegación Gustavo A. Madero, 07738, CDMX

Resumen El presente trabajo proporciona una idea general de las estructura geométricas conocidas como retículas, *Lattice* por su nombre en inglés, y la importancia del algoritmo de reducción de Lestre-Lestralovász (LLL), mediante la creación de una herramienta de software que emplea un motor de animación matemático llamado Manin y PyOpenGL. Mendiante el uso de la herramienta de software, se pretende visualizar de manera gráfica las reticulas y mostrar su contenido sustancial. Por otra parte, en la actualidad existe una nueva rama de la criptografía denominada criptografía post-cuántica. Una de las categorias existentes en es ta criptografía es la basada en reticulas, esta es es importante dentro de la rama de las matemáticas denominada geometría de números, la cual es impresindible para las ciencias computacionales, más en específico en la criptografía post-cuántica.

Palabras claves: algoritmos de reducción de retícula · criptografía basada en retículas · criptografía post-cuántica

Introducción

Para entrar en contexto, en 1994 Peter Shor desarrolló un algoritmo cuántico para la factorización de enteros y el problema del logaritmo discreto, conocido como algoritmo de Shor. Este fue un gran hallazgo en el campo cuántico, dado que hace que prácticamente todas las construcciones criptográficas utilizadas actualmente (como RSA y ECC) sean fácilmente superados por una computadora cuántica.

Se considera que grandes computadoras cuánticas existirán físicamente en los próximos años, estos esquemas de cifrado ya no serán confiables, lo que hace

que los datos cifrados sean comprometidos. Por lo tanto, es de vital importancia construir y analizar nuevos esquemas criptográficos resistentes a los ataques cuánticos.

Estas nuevas construcciones criptográficas deberían ser eficientes y lo suficientemente seguras para ser utilizadas en la práctica y estandarizado por un gran número de años, reemplazando los actuales si es necesario.

Hasta la fecha hay algunos posibles sucesores para ser los próximos esquemas que remplacen a las actuales contrucciones, como lo es el esquema de cifrado **NTRU**, uno de los más seguros y efectivos tanto en computo cuántico como en clásico, desarrollado por Jeffrey Hoffstein, Jill Pipher y Joseph Silverman en los años 90's.

Los ataques más efectivos en NTRU se basan en retículas, es decir, el uso de **Algoritmos de Reducción** (más adelante se precisará este término), cuyo objetivo es encontrar vectores muy cortos dentro de una retícula. En ese sentido se puede aplicar el algoritmo conocido como LLL (1982) de Arjen Lenstra, Hendrik Lenstra y Laszlo Lovasz, el cual se ejecuta en tiempo polinómico y se reconoce como un resultado muy importante en muchas áreas de las matemáticas. Este algoritmo se mejoró con el tiempo.

La mejora más importante se debe a Schnorr (1987), quien introdujo el algoritmo Blockwise-Korkine-Zolotarev, también conocido como algoritmo BKZ, que hasta la fecha es el mejor algoritmo de reducción para ser puesto en práctica.

Hay mucha investigación en esta área, principalmente debido a sus aplicaciones criptográficas, sin embargo han habido pocos acercamientos para reconocer la importancia de la teoría y compresión de estos desarrollos, inclusive dentro del ámbito académico debido a que inicialmente nació como una idea preconcebida sólo como una rama de la matemática.

Estado del Arte

En términos generales, se han construído bastantes alternativas de software con muy buena documentación las cuales pretenden mostrar su importancia de cómo funcionan estas contrucciones criptográficas y su significación, pueden encontrarse empresas como Micrisoft, Maplesoft y Wolfram Research. Las implementaciones existentes del *algoritmo LLL* a presentar son herramientas que ayudan, en primera instancia cuando se enfrenta a trabajar con retículas como lo son:

- **Maple** como la función **IntegerRelations[LLL]**.
- **Mathematica** como la función **LatticeReduce**
- **PARI/GP** como la función **qflll**
- **Magma** como las funciones **LLL** y **LLLGram** (con una matríz de Gram).
- **CrypTool 2** en **Lattice-based cryptography**

Siendo así que todos las anteriores herramientas, con excepción de **CrypTool 2**, que trabajan con retículas muestran un resultado sin visualización alguna, es decir, al ingresar la entrada con vectores requerida sólo dan una salida mostrada

con los vectores reducidos, como coordenadas con números, sin saber cómo se llegaron a estos, pérdiéndose en gran medida del flujo del algorimo y así su significación geométrica. Mientras **CrypTool 2** sólo trabaja en dimensión dos.

Marco de Referencia

Definición 1. Sea $\mathbf{V}$ un $\mathbb{R}$-espacio vectorial de dimensión n, $\beta = \{\mathbf{v}_1, ..., \mathbf{v}_m\}$ una conjunto linealmente independiente. Se define una reticula en $\mathbf{V}$ generada por β como el conjunto de todas las combinaciones lineales con elementos de β con escalares en $\mathbb{Z}$, y es denotada como Λ_β, es decir:

$$\Lambda_\beta = \left\{ \sum_{i=1}^{m} \alpha_i \mathbf{v}_i | \alpha_i \in \mathbb{Z}, \mathbf{v}_i \in \beta \quad \forall i \in \{1, ..., m\} \right\}$$

Se dice que β forma una $\mathbb{Z}$-base para la reícula. En general diferentes bases de $\mathbf{V}$ generarán diferentes reticulas. Aunque si consideramos la matriz de transición entre bases $[T]_{\beta\beta'}$ esta pertenece al grupo lineal general de $\mathbb{Z}$, es decir $GL_m(\mathbb{Z})$, y así las reticulas generadas por estas bases serán isomorfas desde que $[T]_{\beta\beta'}$ induce un isomorfismo entre las dos reticulas.

Definición 2. Sea β una base de $\mathbf{V}$ un $\mathbb{R}$-espacio vectorial de dimensión finita y Λ_β una reticula en $\mathbf{V}$. Un **dominio fundamental** para Λ_β se define como:

$$\mathcal{F}(\Lambda_\beta) = \left\{ \sum_{i=1}^{n} \alpha_i \mathbf{v}_i \quad | \quad 0 \leq \alpha_i < 1 \right\}$$

Los problemas fundamentales y difíciles de las retículas

La conjeturada intratabilidad de tales problemas es fundamental para la construcción de sistemas de cifrado seguros basados en reticulas. Para aplicaciones en dichos sistemas, las retículas se toman en espacios vectoriales (a menudo $\mathbb{Q}^n$) o módulos libres (a menudo $\mathbb{Z}^n$). Considere un espacio normado ($\mathbf{V}$, $||\quad||$) de dimensión n, y Λ una retícula en $\mathbf{V}$, los problemas fundamentales son:

- **El problema del vector más corto (SVP)**:
  Encontrar el vecor no nulo más corto en Λ.
- **El problema del vetor más cercano (CVP)**:
  Dado un vector $\mathbf{t} \in \mathbf{V}$ tal que no esté en Λ, encontrar un vector en Λ más cercano a $\mathbf{t}$.
- **El problema del aproximación al vector más cercano (apprCVP)**
  Dado $\mathbf{t} \in \mathbf{V}$, encontrar un vector $\mathbf{v} \in \Lambda$ tal que $||\mathbf{v} - \mathbf{t}||$ es pequeño. Es decir: $||\mathbf{v} - \mathbf{t}|| \leq k \min_{\mathbf{w} \in L} ||\mathbf{w} - \mathbf{t}||$ Para una constante k pequeña.

Los problemas de retículas, **SVP** y **CVP**, se han estudiado intensamente, tanto problemas en matemáticas puras y aplicadas, como para criptografía.

Reducción de una Retícula: El nombre dado al problema práctico de resolver SVP y CVP, o más generalmente de encontrar vectores razonablemente cortos y bases *buenas* o más convenientes.

Encontrar Bases Razonablemente "Buenas".

La idea de la reducción de la base es cambiar una base β de una retícula Λ en una base más corta β' de tal manera que permanezca inalterada. Para hacer esto, podemos usar las siguientes operaciones:

- 1. Intercambiando dos vectores de la base. Como el intercambio cambia solo el orden de los vectores en la base, es trivial que Λ no se vea afectada.
- 2. Reemplazar v_j por $-v_j$. Es trivial que Λ no se ve afectada.
- 3. Sumar (o restar) a un vector v_j una combinación lineal y discreta de los otros vectores de la base. La retícula no se ve afectada porque si tomamos un vector arbitrario v que pertenece a Λ podemos expresarlo como una combinación discreta de los vectores de la base:

$$w = \sum_{i=1}^{n} \alpha_i \mathbf{v}_i$$

Y si luego reemplazamos v_j por una combinación discreta de los otros vectores de la base

$$\mathbf{v_j} \leftarrow \mathbf{v_j} + \sum_{i=1, i\neq j}^{n} y_i \mathbf{v}_i$$

Todavía se puede expresar w como una combinación discreta de los vectores de la nueva base:

$$w = \sum_{i=1, i\neq j}^{n} \alpha_i \mathbf{v}_i + \alpha_j (v_j - \sum_{i=1, i\neq j}^{n} y_i \mathbf{v}_i)$$

De manera similar, podemos mostrar que si w no pertenece a Λ, entonces no podemos expresarlo con una combinación discreta de la nueva base. Se deduce que las dos $\mathbb{Z}$-bases generan exactamente la misma reticula. La reducción de una retícula se puede utilizar para resolver el problema de vector más corto en el sentido de que el vector más corto de la $\mathbb{Z}$-base es tal vector dentro ella, cabe resaltar de que este no es único [1].

Definición 4. Se dice que una retícula es degenerada si la cardinalidad de su $\mathbb{Z} - base$ es menor a la del espacio, y no degenerada si es la carndinalidad de su $\mathbb{Z}-$base es igual a la del espacio.

Las condiciones de tamaño y de pseudo-ortogonalidad.

Suponga que tiene un conjunto linealmente independiente de vectores $\{\mathbf{v_1}, ..., \mathbf{v_n}\}$ y después del proceso de ortogonalizaión de Gram-Schmidth se obtiene un conjunto $\{\mathbf{v*_1}, ..., \mathbf{v*_n}\}$. Si algún coeficiente en el proceso de Gram-Schmidt satisface:

$$\frac{|\mathbf{v_i} \cdot \mathbf{v*_j}|}{||\mathbf{v*_j}||^2} > \frac{1}{2}$$

Luego reemplazando v_i por $v_i - av_j$ con un a apropiado en $\mathbb{Z}$ hace que el coeficiente sea más pequeño. Decimos que una base satisface la **Condición de Tamaño** si:

$$\frac{|\mathbf{v_i} \cdot \mathbf{v*_j}|}{||\mathbf{v*_j}||^2} \leq \frac{1}{2} \quad \forall i < j$$

Para equilibrar esto, queremos que los vectores base sean lo más ortogonales entre sí, por lo que imponemos la **Condición de Pseudo-Ortogonalidad**:

$$||\mathbf{v*_{i+1}}|| \geq \frac{\sqrt{3}}{2}||\mathbf{v*_i}||$$

Teorema 1: Hermite
En cada retícula existe una base que satisface tanto la Condición de tamaño como la Condición de pseudo-ortogonalidad.
Demostración.[5]

Desafortunadamente, los algoritmos más conocidos para encontrar esa base son exponenciales en la dimensión. Entonces cambiamos la Condición de pseudo-ortogonalidad a una menos escritcta, la **Condición de Lovász**:

$$||\mathbf{v*_{i+1}}|| \geq \sqrt{\frac{3}{4} - \frac{|\mathbf{v_{i+1}} \cdot \mathbf{v*_i}|}{||\mathbf{v_i}||^2}}||\mathbf{v*_i}||$$

Lo único que dice esta condición es que la proyección de $\mathbf{v_{i+1}}$ dentro del complemento ortogonal del subespacio generado por los vectores $\mathbf{v_1}, ..., \mathbf{v_{i-1}}$ es mayor o igual a tres cuartos de la proyección de $\mathbf{v_i}$ dentro del complemento ortogonal de del subespacio generado por los vectores $\mathbf{v_1}, ..., \mathbf{v_{i-1}}$. Que de hecho resulta ser una generalización de la condición de pseudo-ortogonalidad.

Teorema 2: (Lenstra, Lenstra, Lovász)
Existe un algoritmo de tiempo polinómico que encuentra una base para que Λ satisfaga tanto la Condición de tamaño como la Condición de Lovász. Dichas bases se denominan bases reducidas de LLL. **Demostración.**[5]

Reducción de Base LLL.

Ya que se busca que el usuario tenga un interacción visual de como funciona el algoritmo LLL es necesario saber como es que se visualiza el resultado arrojado por el LLL y se mezcla junto con la gráfica de la retícula. El algoritmo de

reducción LLL con el que se va a trabajar es capaz de resolver el problema del vector más corto en dimesiones superiores casi arbitrarias.

Implementacion.
Sea $\beta = \{b_1, b_2, \ldots, b_m\}$ un base donde $b_i \in \mathbb{R}^n$ y sea $c \in \mathbb{R}$ tal que $c \geq 4/3$. Ahora para cada $i = 1, \ldots, m$ sea $u_i = (0, 0, \ldots, 0)$ excepto que en la entrada $i - esima$, $u_{i_i} = 1$, definimos que b_i^* sea igual a b_i.
Así pues para cada $j = 1, \ldots, i-1$ sea u_{ij} (la $j - esima$ entrada de u_i) tal que $u_{ij} = (b_i b_j^*)/(b_j^* b_j^*)$, luego finalmente sea $b_i^* = b_i^* - u_{ij} b_j^*$.

Para la siguiente parte empezando desde $i = 1$, si $\|b_i^*\|^2 \leq c\|b_{i+1}^*\|^2$ entonces $\{b_1, \ldots, b_{i+1}\}$ es c-reducido y continuamos con $i + 1$.

De lo contrario definimos $b_{i+1}^* = b_{i+1}^* + u_{i+1_i} b_i^*$, luego se define a $u_{ii} = (b_i b_{i+1}^*)/(b_{i+1}^* b_{i+1}^*)$ y hacemos que $u_{i_{i+1}}$ sea igual 1 y de igual forma con $u_{i+1_i} = 1$, ademas también que $u_{i+1_{i+1}} = 0$.

Definiendo $b_i^* = b_i^* - u_{ii} b_{i+1}^*$, intercambiamos a u_i con u_{i+1}, b_i^* con b_{i+1}^* y b_i con b_{i+1} en ese orden. Finalmente para $k = i+2$ hasta $k = m$ se hace lo siguiente: definimos $u_{ki} = (b_k b_i^*)/(b_i^* b_i^*)$ y de igual forma $u_{ki+1} = (b_k b_{i+1}^*)/(b_{i+1}^* b_{i+1}^*)$.

Ahora si $|u_{i+1i}| > 1/2$ entonces aplicamos lo de arriba para $i + 1$ y hacemos que $i = max(i-1, 1)$. Como como subrutina hacemos que mientras $j = i-1 > 0$ se define $b_i = b_i - \lfloor u_{ij} \rfloor b_j$ y que $u_i = u_i - \lfloor u_{ij} \rfloor u_j$ y terminamos con $j = j - 1$.

Entonces al final de estas rutinas se debería de obtener un base casi ortogonal $\{b_1^*, b_2^*, \ldots, b_m^*\}$ y vectores de proyección $\{u_1, u_2, \ldots, u_m\}$ tales que $b_i^* \in \mathbb{R}^n$ y $u_i \in \mathbb{R}^m$.

Para llevar acabo las graficas de retículas de dimensiones superiores, es necesario hacer un esquema teórico de como se va a graficar.

Definición 3. Dado un vector pivote $c \in \mathbb{R}^n$ y una base $\beta = \{v, v_2, \ldots, v_n\}$, se dice que un **tarugo** T_c^β es un conjunto de 2^n vectores de $\mathbb{R}^n$, tales que

$$T_c^\beta = \{c + a_1 v_1 + a_2 v_2 + \ldots a_n v_n : a_i \in \{0, 1\} \text{ y } v_i \in \beta\}.$$

Observación. Dada una base $\beta \subset \mathbb{R}^n$ se puede formar una retícula Λ, y por la definición de T_c^β con $c \in \mathbb{R}^n$ basta que $c \in \Lambda$ para que así mismo $T_c^\beta \subset \Lambda$. Entonces el esquema que se esta proponiendo para que un programa construya una retícula cualesquiera apartir de una base β es que gráfique T_c^β para cada $c \in \Lambda$ deseado. No es difícil poder identificar cual debe de ser la forma de $c \in \Lambda$ ya que por definicón de Λ si un $x \in \Lambda$, entonces se cumple que existen $k_1, k_2, \ldots, k_n \in \mathbb{Z}$ tales que

$$x = k_1 v_1 + k_2 v_2 + \ldots + k_n v_n$$

Por tanto es trivial decidir de que forma tiene que ser el vector pivote c del tarugo T_c^β. Este esquema de construcción ofrece ciertas ventajas, tales como poder gráficar regiones especificas de Λ o en su defecto evitar dibjar ciertas regiones de Λ. Así pues se construye una herramienta que es facíl de aplicar y permite un acceso para poder visualizar retículas de dimensiones superiores sin problema alguno.

Propuesta de Solución

Lo que se esta proponiendo es una herramienta de visualización que no solo implementa el algoritmo LLL sino que también lleve acabo la grafica de la retícula dada su base en un espacio euclideano. Así el usuario tendra acceso a una herramienta que visualiza retículas en los espacios euclideanos de dimesiones superiores a tres dimensiones. La herramienta de visualización propuesta se compone de dos partes, ambas unidas por una **GUI** que permite escoger entre ambas opcciones.

1. La primera parte (en la dimensión dos y tres) sobre el motor de animación denominado **manim** el cual está compuesto por:

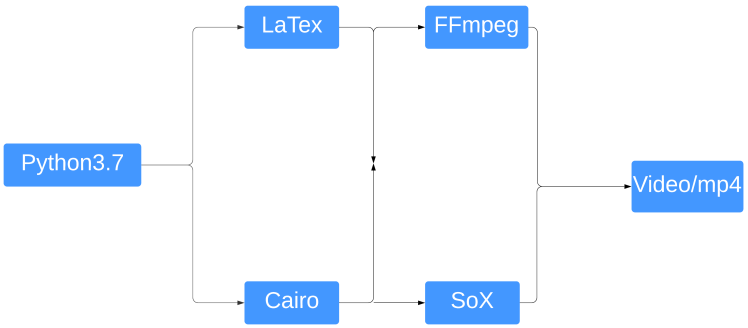


Figura 1: Diagrama de funcionamiento de manim.

Donde el texto impreso en pantalla está escrito en LaTex, Cairo es una librería que ayuda con algunos elementos gráficos ya construídos y FFmpeg renderiza el resultado final como imagen o video, finalmente todas estas instrucciones se escriben en el lenguaje de programación python3.7. En el siguiente diagrama se explica las opciones.

Se crearon dos programas fundamentales en esta primera parte, el principal puede graficar tanto retículas degeneradas como no degeneradas, llendo con espacios de dimensión dos, tres y cuatro como opciones.

El segundo tiene la posibilidad de graficar un dominio fundamental de cada una de la retícua posible y en el caso de las retículas no degeneradas aplicar el algoritmo de reducción LLL en dimensión dos y tres. Para ello se usaron una serie de módulos propios de manim y librerías externas tales como las que aparecen a continuación.

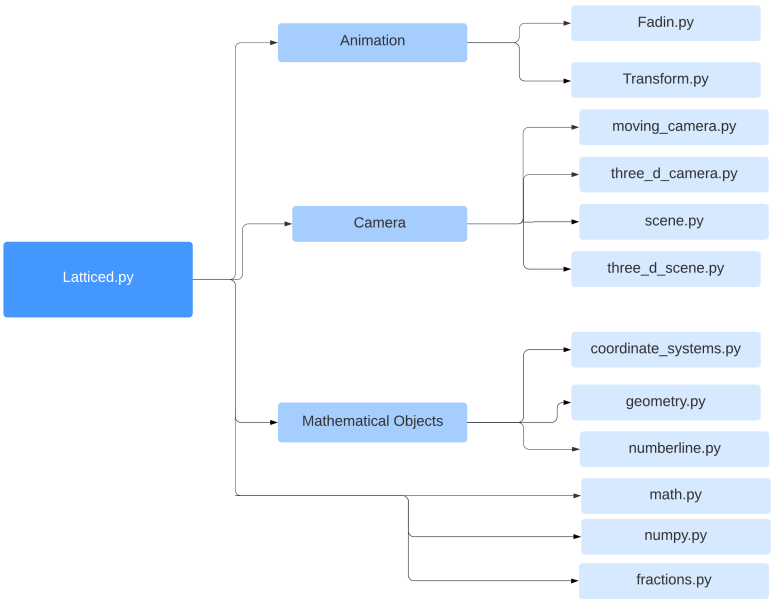


Figura 2: Diagrama de módulos escenciales de la herramienta de visualización con **manim**.

2. La segunda parte, se desarrollo para poder gráficar en dimensiones cuatro y cinco. Esto se construyó usando nuevamente python3.7 utilizando una biblioteca llamada PyOpenGL. Se utilizo dicha biblioteca ya que proporciona gráficos de buen detalle y su conexión a la GUI no presenta problema alguno.

Esto se debe a que en PyOpenGL solo es necesario definir una sola función, sea f tal función, la cual gráfica en cuanto es llamada, dicha función se define para que este en espera de datos proporcionados por el usaurio. Una vez que tiene los datos manda a llamar a las funciones encargadas de dibujar a los tarugos que funcionan usando el metodo de triangulización y así mismo se dibuja la retícula.

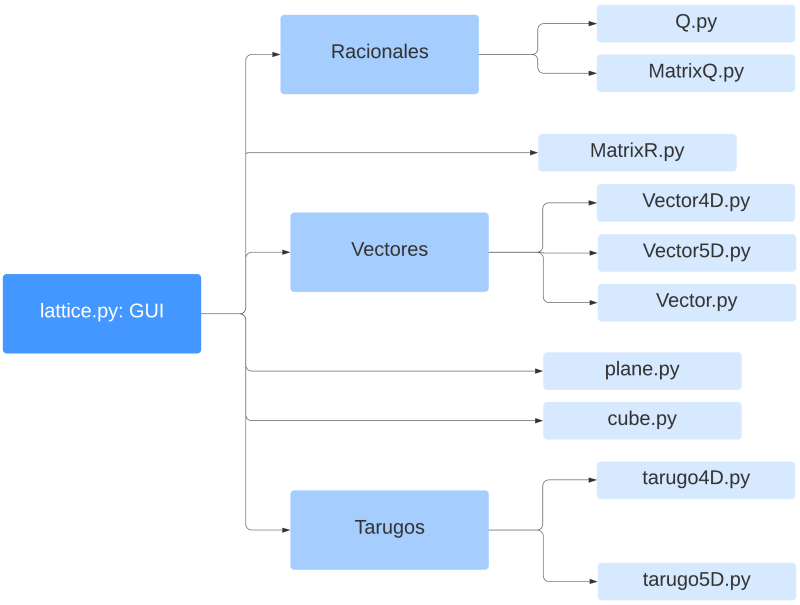


Figura 3: Diagrama de módulos requeridos de la herramienta de visualización con OpenGl.

Ahora, f se conecta a la GUI mediante un boton que en cuanto es presionado manda a llamar a a f y esta realiza la gráfica. Por tanto para poder llamar a f adecuadamente solo es necesario que el usaurio ingrese los datos de las bases y presione el boton conectado a f.

Ya se mencionó que la retícula se dibuja mediante tarugos, sin embargo cabe mencionar que en el la renderización dada por PyOpenGL los tarugos de dimensiones cuatro y cinco debe ser cortados para que así se puedan viasualizar y no causen mucho ruido visual. Cuando se dice que son cortados, solo se dibuja una celda por tarugo, es decir, como si solo dibujaramos una cara de un cubo y omitieramos las demas, con el fin de poder visualizar y ahorra poder de computo. Entonces una retícula de dimensión cuatro se visualiza por celdas especificas de teseractos (cubos de dimensión cuatro) deformados por tranformaciones lineales determinadas por la $\mathbb{Z}$-base de la retícula la cual es introducida por parte del usuario dentro de la GUI como entradas racionales de vectores.

Lo mismo ocurre en dimensión cinco, la retícula es visualizada por cortes a los penteractos (cubos de dimensión cinco) y lo sobrante de los cortes resultan ser teseractos deformados por transformaciones lineales determinadas por la $\mathbb{Z}$-base de la retícula.

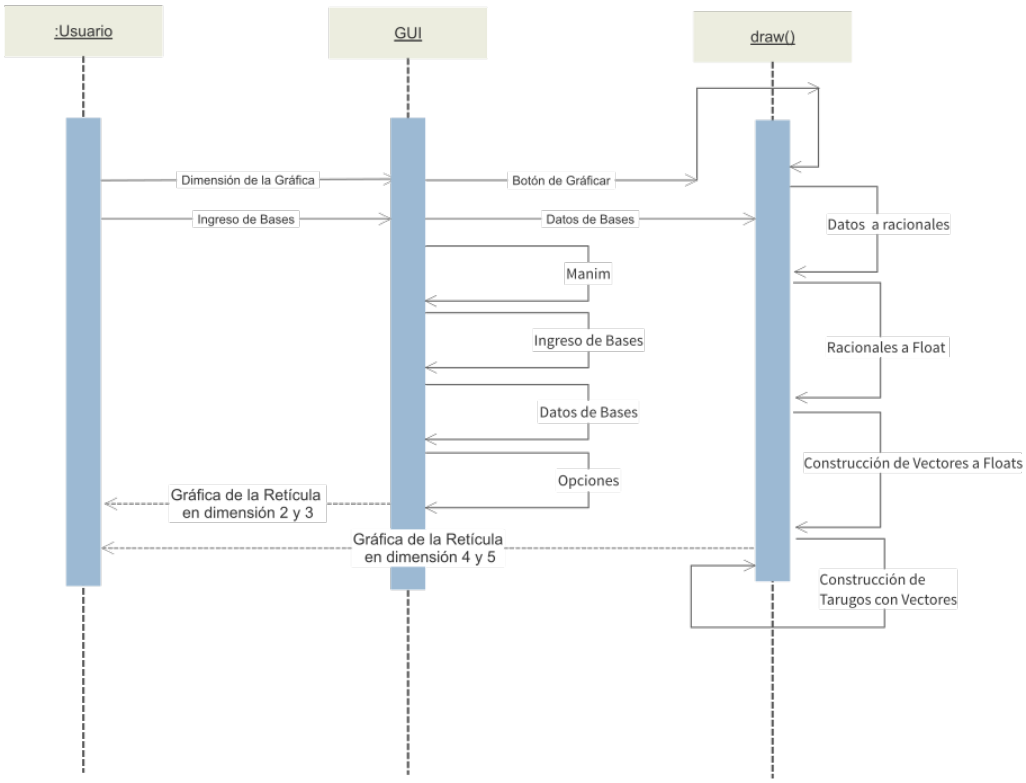


Figura 4: Diagrama de funcionamiento de la herramienta de visualización con OpenGL, manim y la GUI.

Pruebas

Para la siguiente serie de pruebas es necesario contar con los requerimientos citados en [7] que principalmente hace la mayor importancia en cuanto a la estructura algebraica, en la primera parte del proyecto, el uso de numpy 1.16.4. Los siguientes datos son las $\mathbb{Z}$-bases propuestas.

Dimensión 2.

- *Retícula degenerada*: $v_1 = (0.2, 2.1)$.
- *Retícula no degenerada* : $v_1 = (12, 3)$, $v_2 = (4, 7)$.

Dimensión 3.

- *Retícula 1-degenerada*: $v_1 = (2.3, 1.66, 5.4)$.
- *Retícula 2-degenerada*: $v_1 = (0.5, 1, 2.3)$, $v_2 = (0, 3, 1)$.
- *Retícula no degenerada*: $v_1 = (13, 2, 5)$, $v_2 = (12, 5, 6)$, $v_3 = (10, 3, 7)$.

Para el Dominio Fundamental.

- *Retícula no degenerada bidimensional*: $v_1 = (0.4, 1)$, $v_2 = (2.2, 0.3)$.
- *Retícula no degenerada tridimensional*:
  $v_1 = (0.5, 2, 1)$, $v_2 = (1, 3, 0.4)$, $v_3 = (3, 1, 0.6)$.

Para un conjunto Linealmente Dependiente aplicado al LLL.

- *Retícula no degenerada en bidimensional*: $v_1 = (1, 1)$, $v_2 = (2, 2)$.
- *Retícula no degenerada en tridimensional*:
  $v_1 =(1, 1, 1)$, $v_2 = (2, 3, 4)$, $v_3 = (\ 2, 2, 2)$.

Para la segunda parte se utilizaron nuevos módulos en python3.7 los cuales tienen como objetivo el manejo de los tipos de datos racionales y su implementación con matrices para incorporarlas a la herramienta mediante OpenGL y el GUI, para ello se crearon los módulos referenciados en la **Figura 4**, los cuales se encuentran en el repositorio del proyecto [8].

Dimensión 4.

- *Retícula no degenerada*:
  $v_1 = (1, 0, 0, 0)$, $v_2 = (0, 2, 0, 0)$, $v_3 = (0, 0, 3, 0)$, $v_4 = (0, 0, 0, 4)$.

Dimensión 5.

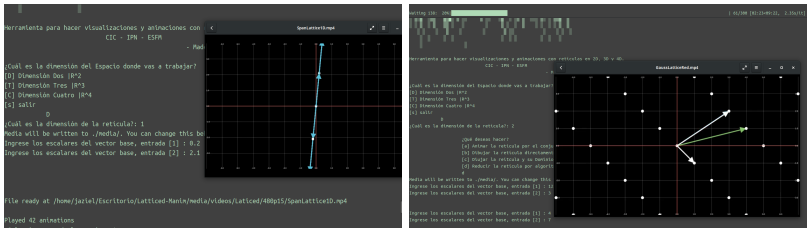
- *Retícula no degenerada*:
  $v_1 = (1/2, 0, 0, 0, 0)$, $v_2 = (0, 1, 0, 0, 0)$, $v_3 = (0, 0, 3/2, 0, 0)$,
  $v_4 = (0, 0, 0, 2, 0)$, $v_5 = (0, 0, 0, 0, 5/2)$

Resultados

La herramienta nos da resultados correctos haciendose valer de pruebas numéricas con numpy antes de introducir este método de visualización con tarugos y el algorimo LLL.

En la **Figura 5** se observa que para el inciso (a) el vector $\mathbb{Z}-$base es $v_1 = (0.2, 2.1)$. Mientras que para en inciso (b) su $\mathbb{Z}$-base está formada por los vectores $v_1 = (12, 3)$ y $v_2 = (4, 7)$ a la cual se le aplica el algorimo de reducción LLL.

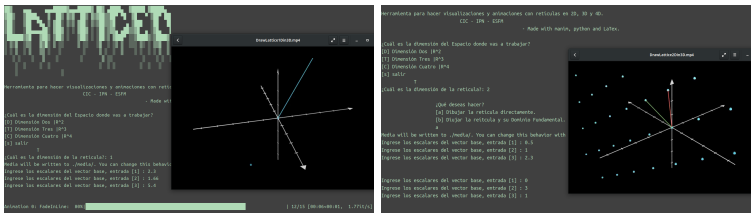
Se observa que en la segunda dimensión la gráfica de la retícula degenerada por medio de escalamiento del vector base y posteriormente insertando puntos es correcta, así como la implementación de tarugos para la retícula no degenerada cuyo algoritmo de reducción fue empleado al conjuto l.i antes mencionado.



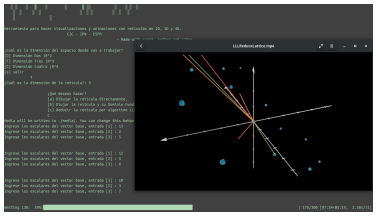
(a) Retícula degenerada (b) Retícula no degenerada

Figura 5: Retículas en el espacio bidimensional.

En la **Figura 6** se observa que para el inciso (a) el vector $\mathbb{Z}-$base es $v_1 = (2.3, 1.66, 5.4)$. Mientras que para en inciso (b) su $\mathbb{Z}$-base está formada por los vectores $v_1 = (0.5, 1, 2.3)$, $v_2 = (0, 3, 1)$. Finalmente para el inciso (c) está dada por los vectores $v_1 = (13, 2, 5)$, $v_2 = (12, 5, 6)$, $v_3 = (10, 3, 7)$.



(a) Retícula 1-degenerada (b) Retícula 2-degenerada

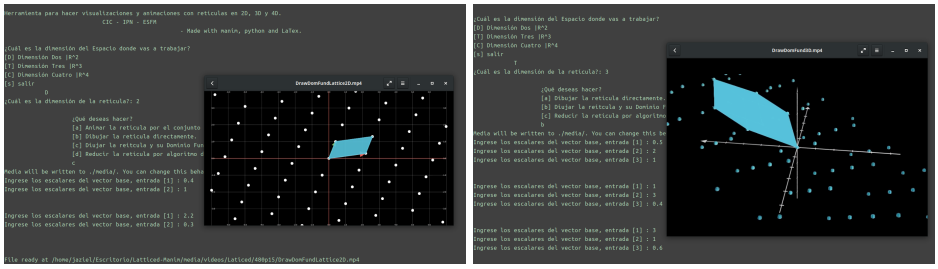


(c) Reticula no degenerada

Figura 6: Retículas en el espacio tridimensional.

Para la retícula 1-degenerada y 2-degenerada en el espacio tridimensional la implementación de tarugos es funcional. Ahora bien se aplica el algoritmo de reducción LLL a la retícula determinada por la $\mathbb{Z}$-base antes mencionada y cuyo resultado se muestra en la **Figura 6**.

En la **Figura 7** se observa que para el inciso (a) el conjunto de vectores forma parte de una $\mathbb{Z}-$base: $v_1 = (0.4, 1)$, $v_2 = (2.2, 0.3)$. Mientras que para en inciso (b) su $\mathbb{Z}$-base está formada por los vectores $v_1 = (0.5, 2, 1)$, $v_2 = (1, 3, 0.4)$, $v_3 = (3, 1, 0.6)$.

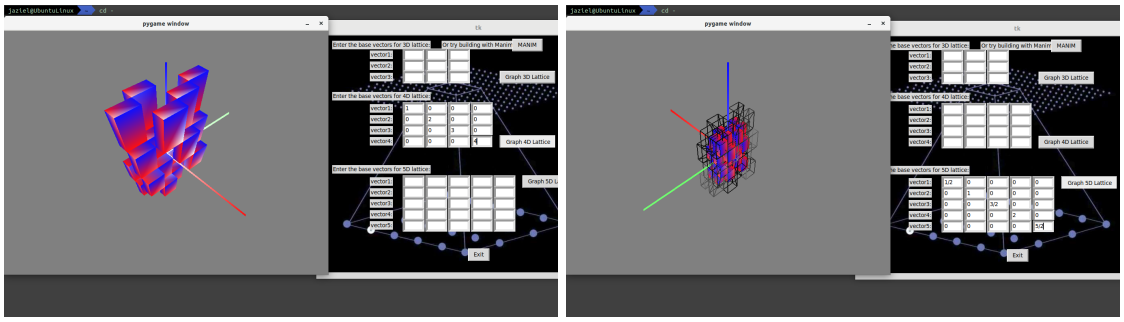


(a) Dominio Fundamental bidimensional (b) Dominio Fundamental tridimensional

Figura 7: Dominio Fundamental.

Ahora bien, para la parte de dimensión cuatro y cinco cada $\mathbb{Z}$-base es introducida en la GUI como enteros o racionales cuyo resultado es el siguiente. En la **Figura 8** se observa que para el inciso (a) la $\mathbb{Z}-$base está determinada por: $v_1 = (1, 0, 0, 0)$, $v_2 = (0, 2, 0, 0)$, $v_3 = (0, 0, 3, 0)$, $v_4 = (0, 0, 0, 4)$.

Mientras que para en inciso (b) su $\mathbb{Z}$-base está formada por los vectores: $v_1 = (1/2, 0, 0, 0, 0)$, $v_2 = (0, 1, 0, 0, 0)$, $v_3 = (0, 0, 3/2, 0, 0)$, $v_4 = (0, 0, 0, 2, 0)$, $v_5 = (0, 0, 0, 0, 5/2)$.

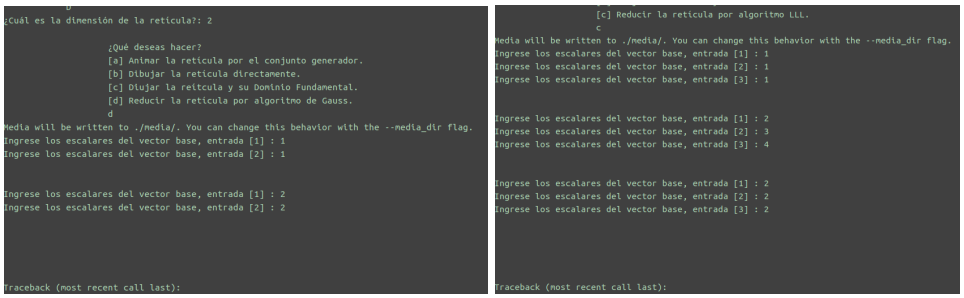


(a) Retícula no degenerada tetradimensional (b) Retícula no degenerada pentadimensional

Figura 8: Dimensión 4 y 5.

En la **Figura 9** se observa que para el inciso (a) el conjunto de vectores linealmente dependiente formado por los vectores: $v_1 = (1, 1)$, y $v_2 = (2, 2)$.

Asimismo para en inciso (b) se tiene un conjunto de vectores linealmente dependiente en el espacio tridimensional formado por:
$v_1 =(1, 1, 1)$, $v_2 = (2, 3, 4)$, y $v_3 = (\ 2, 2, 2)$.



(a) Conjunto de vectores l.d. en $\mathbb{R}^2$ (b) Conjunto de vectores l. d. en $\mathbb{R}^3$

Figura 9: Para un conjunto Linealmente Dependiente aplicado al LLL

Se observa que en el caso de ingresar un conjunto lineal ente dependiente de vectores en el espacio bidimiensional o tridimensional y alicarle el algoritmo LLL arroja un requerimiento no funcional. Es decir tenemos por parte del módulo *fractions.py* de la Figura 2, que forma parte de los módulos de la primera parte, que en el gestor de errores se tiene un *ZeroDivisionError: division by zero*.

Ahora bien, se hará la comparativa con el estado del arte, más especificamente con Wolphram y su método LatticeReduce, y verificar que el algoritmo implementado aquí sea correcto, para lo cual las $\mathbb{Z}$-bases a reducir son las mismas en la sección de pruebas para retículas no degeneradas tanto bidimensionales como tridimensionales.

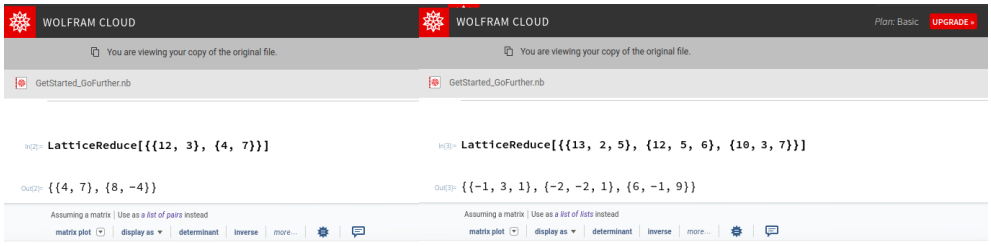


Figura 10: Reducción de retículas por medio del algoritmo LLL en Wolfram.

Ahora bien, se extrae la pieza de código que realiza la reducción de la retícula por medio del algoritmo LLL alojada dentro de la herramienta y arroja el siguiente resultado.

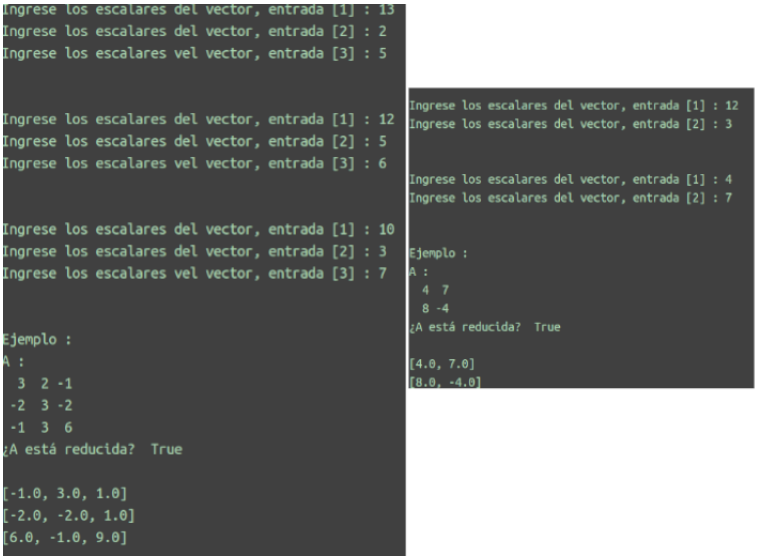


Figura 11: Reducción de retículas por medio del algoritmo LLL por medio de la implementación alojada dentro de la herramienta.

Y en efecto coinciden ambos resultados vistos como vectores columna.

Discusión

La implementación de la teoría de tarugos para poder optimizar recursos a la hora de renderizar es en efecto, eficiente y, arroja resultados correctos tanto para retículas degenerados como no degeneradas. Para la implementación del algorimo LLL dentro de una retícula de dimensión dos y tres no hubo mayor difícultad, siempre y cuando los vectores introducidos no fueran linealmente dependientes la herramienta funciona de manera adecuada. De no ser así arroja un requerimiento no funcional, ya que dentro del algoritmo LLL se efectuaría una división por cero y se notifica con un *try-except block*. Aun no se implementa una manera adecuada de manejar los errores sin salirse de la interfáz como en el caso que se le inserte un conjunto linealmente dependiente de vectores.

Cabe resaltar que entre más densa sea la retícula, y mayor número de elementos dentro del espacio de visualización, más le costará a la herramienta, en términos de tiempo, terminar de renderizar, ya que tendría que renderizar más

objetos en pantalla, y ello también dependerá qué calidad de imagen se requiera, baja, media o alta. Ahora bien esto ha presentado un problema con la dimensión cuatro y cinco, ya que para graficar correcamente el número de aristas y vértices incrementa notoriamente a las dimensiones anteriores, esto implica que el tiempo para la representación fianal mucho mayor por lo que se optó a representar una retícula con OpenGL como los vértices de hipercubos y penteractos, respectivamente, con ello se resolvio el problema intrínseco del metor de animación manim.

Se desarroló un método basado en tarugos para graficar el dominio fundametal de cada retícula, esto simpificará mucho pasar entre dimensiones sin ningún problema. Finalmente a la hora de implementar el arloritmo LLL para reducir una retícula hasta las dimensiones en las que se ha trabajado no se ha encontrado ningún problema, se reitera, menos que el conjunto de vectores entrada sea linealmente dependiente se genra un error en la herramienta.

Finalmente la implementación de una GUI con el módulo tinker es de mayor ayuda a la hora de unificar ambas partes, además también ayuda a identificar las partes de la herramienta, así como sus opciones.

Conclusiones y Trabajos Futuros.

Se ha mostrado aqui una manera eficiente de graficar retículas y más aún, politopos de diferentes dimensiones, haciendo uso de la teoría de tarugos. Esto es importante resaltar, ya que no existía una alernativa de software para graficar retículas de distintas dimensiones y que también se implementen algoritmos de reducción.

La teoría detrás de la criptografía basada en retículas es de ende matemático e inspirado por el reino geométrico y por ello existe una necesidad mayor de visualizar su fundamento. Y en este caso fue de gran ayuda para construir la misma herramienta.

También se ha explorado dimensiones mayores, que en el caso de la cuatro y cinco, por crecer exponencialmente el número de elementos tarda bastante en renderizar la totalidad de ellos, siendo así, el motor de animación ahora representa una desventaja a la hora de graficar a comparción de pyOpenGL. Pero al final el resultado es bastante uniforme presentando una herraienta unificada. Como trabajo a futuro es implementar el algoritmo LLL a dimensión cuatro y cinco de una manera visual y óptima, como es el caso de dimensión dos y tres. Quizá cambiar a un lenguaje compilado en lugar de uno interpretado para optimizar el tiempo de renderización. Así como una mejor forma para manejar errores de división por cero, asimismo sugerencias al usuario y mejorar la GUI.
Como comentario final, esta pieza de software tiene como objetivo ilustrar a todo aquél que estudia criptografía post-cuántica cual es el problema matemático de fondo, entender de qué forma es totalmente diferente a cualquier otro dentro de la criptografía clásica.

Referencias

1. Jose Silverman Jeffrey Hoffstein, Jill Pip-her.An Introduction to Mathematical Criptography. Springer; 2 edition, 2018.
2. D. Simon. Selected applications of LLL in number theory. Bosma, Wieb. "4.LLL"(PDF). Lecture notes, 2010.
3. Hendrik W. Lenstra, Jr. Lattices. In Algorithmic number theory: lattices, number fields, curves and cryptography Math. Sci. Res. Inst. Publ. 127-181 Cambridge Univ. Press Cambridge, 2008.
4. Siegel, C.L., Lectures on the geometry of numbers. Springer-Verlag, Berlin,1989.
5. Lenstra, A., Lenstra, H., Lovasz, L., Factoring polynomials with rational coefficients, Mathematische Ann. 261 (1982), 513-534.
6. Bernstein, D. J., Buchmann, J., Dahmén, E. (2009). Post-quantum cryptography. Berlin: Springer.
7. Grant Sanderson. (2019). www.3blue1brown.com . https://github.com/3b1b . https://github.com/3b1b/manim/blob/master/requirements.txt .
8. Jaziel Flores. (2020). https://github.com/JazzzFM/Latticed-Manim .