ISSN 2395-8618

Herramienta de software para la visualización de la criptografía basada en retículas

Jaziel D. Flores Rodríguez, Miguel E. Pérez Ibarra, Fernando Q. Valencia Rodríguez, Alfonso F. De Abiega L'Eglisse and Gina Gallegos García

***Resumen*—En la actualidad la criptografía post-cuántica puede ser estudiada a través de una categorización que marca la diferencia entre la criptografía basada en isogeneas de curvas elípticas, ecuaciones de múltiples variables y retículas, por mencionar algunas. Desde el punto de vista de la geometría de números, las retículas llamadas *Lattices* por su nombre en inglés, son estructuras geométricas que también pueden verse como el conjunto de todas las combinaciones enteras de las denominadas $\mathbb{Z}$-bases que son conjuntos de vectores linealmente independientes. Con base en ello, en este trabajo se presenta el diseño de una herramienta de *software*, que emplea un motor de animación matemático llamado *Manim* y *PyOpenGL*, la plataforma interoperable de Python que permite unir OpenGL con otras interfaces de programación de aplicaciones. Su uso permitirá visualizar, de manera gráfica, las retículas además de mostrar su contenido sustancial.**

***Palabras clave*—Algoritmos de reducción de retículas, criptografía, desarrollo de software.**

Software Tool for Visualization of Lattice-based Cryptography

***Abstract*—Currently, post-quantum cryptography can be studied through a categorization that distinguishes between cryptography based on isogenies of elliptic curves, multivariate equations, and lattices, among others. From the perspective of number theory, lattices, known as "Latticesïn English, are geometric structures that can also be seen as the set of all integer combinations of so-called $\mathbb{Z}$-bases, which are sets of linearly independent vectors. Based on this, this work presents the design of a software tool that utilizes a mathematical animation engine called Manim and PyOpenGL, the interoperable Python platform that integrates OpenGL with other application programming interfaces. Its use will allow graphical visualization of lattices and display their substantive content.**

***Index Terms*—Lattice reduction algorithms, cryptography, software development.**

Manuscript received on 06/07/2022, accepted for publication on 08/12/2023.

J.D. Flores Rodríguez and M.E. Pérez Ibarra are with the Instituto Politécnico Nacional. Escuela Superior de Física y Matemáticas, CDMX, México (jazzzfm@protonmail.com, supersluky@gmail.com).

F.Q. Valencia Rodríguez is with the Instituto Politécnico Nacional. Escuela Superior de Computación, CDMX, México (quetzalfir@hotmail.com).

A.F. De Abiega L'Eglisse is with the Instituto Politécnico Nacional. Escuela Superior de Ingeniería Mecánica y Eléctrica Unidad Culhuacan , CDMX, México (alfonso.deabiega@tec.mx).

Gina Gallegos García is with the Instituto Politécnico Nacional. Centro de Investigación en Computación, CDMX, México (ggallegos@cic.ipn.mx).

I. Introducción

En 1994 Peter Shor desarrolló un algoritmo cuántico para resolver el problema de la factorización de un número y el problema del logaritmo discreto, el cual se conoció como algoritmo de Shor [1]. Este desarrollo se condsideró de gran importancia dado que las construcciones criptográficas de llave pública, entre las que destacan, el esquema de RSA y ElGamal, utilizadas hasta la fecha, serían vulneradas llegada la existencia de una computadora cuántica.

Lo anterior, ha generado que durante los últimos años, exista una oleada de investigación científica enfocada en el diseño y la construcción de esquemas criptográficos capaces de resistir ataques efectuados, tanto por computadoras clásicas, como por computadoras cuánticas. Dichos esquemas reciben el nombre de esquemas post-cuánticos.

Dependiendo de la funcionalidad y del problema en el que basan su seguridad, los esquemas post-cuánticos se clasifican en 5 tipos: isogeneas de curvas elípticas, polinomios cuadráticos multivariables, códigos, hashes y esquemas basados en retículas. Específicamente hablando de los esquemas que basan su funcionalidad en retículas, el algoritmo más conocido es el esquema de cifrado de *NTRU*, siendo este uno de los mas seguros y efectivos tanto en computo cuántico como en clásico [2].

Los ataques que se condieran más efectivos hacia el esquema de *NTRU*, hacen uso de *algoritmos de reducción*, que se enfocan en encontrar vectores muy cortos dentro de un retícula. Un algoritmo de este tipo, es el llamado *LLL* el cual se ejecuta en tiempo polinómico [3]. Años después, vinieron mejoras de algoritmos de redución, entre las que destaca el algoritmo *BKZ* [4], que hasta la fecha se considera como el mejor algoritmo de reducción que puede ser puesto en la práctica.

Con base en lo anterior, en este trabajo se presenta el diseño de una herramienta de *software* para la visualización de la funcionalidad que poseen criptografía basada en retículas, muestra la funcionalidad de las construcciones criptográficas, antes mencionadas. El resto del documento, se organiza de la siguiente forma: en la Sección II se pueden apreciar los trabajos relacionados con la investigación realizada en este documento. La Sección III dicta la teoría matemática detrás de los problemas existentes en las retículas, mostrando sus definiciones y teoremas. La Sección IV muestra el diseño de la herramienta capaz de animar retículas de dos, tres,

ISSN 2395-8618

cuatro y cinco dimensiones. La Sección V muestra las pruebas que fueron realizadas a la herramienta de *software*, también presenta los resultados obtenidos. La Sección VI presenta la discusión de dichos resultados. Por último, la Sección VII señala las conclusiones y el trabajo a futuro que se desprende de esta herramienta de *software* para visualización de la criptografía basada en retículas.

II. Estado del arte

A la fecha, se han construído herramientas de *software* que tienen la tarea de mostrar la funcionalidad de las diferentes contrucciones criptográficas.
Específicamente hablando de herramientas que implementen algoritmos de reducción como el *Algoritmo LLL* existen: *Cryptography Tools* [5], *Maple* [6], que tiene la función *IntegerRelations[LLL]*. *Mathematica* [7], que implementa la función *LatticeReduce*. PARI/GP [8], con la función *qflll*. *Magma* [9], que implementa las funciones *LLL* y *LLLGram*, haciendo uso de una matriz de Gram. Y *CrypTool 2* [10], con *Lattice-based cryptography*. Sin embargo, la mayoría de estas herramientas muestran sus resultados sin visualizar su funcionalidad paso a paso. Es decir, al ingresar vectores como una entrada requerida, estas herramientas solamente dan una salida mostrada con los vectores reducidos y como coordenadas con números. Esto, sin mostrar el detalle que indique cómo se llegó a ellos, lo cual genera desconocimiento en gran medida del flujo de los datos del algoritmo y como consecuencia, en su significado geométrico [11]. Adicionalmente a ello, *CrypTool 2* [10] permite trabajar unicamente con dimensión dos.

III. Marco teórico

En esta sección se presentan las definiciones, que se consideran más importantes para el diseño de la herramienta de *software* para visualización.

Definición 1.

Sea $\mathbf{V}$ un $\mathbb{R}$-espacio vectorial de dimensión n, $\beta = \{\mathbf{v}_1, ..., \mathbf{v}_m\}$ un conjunto linealmente independiente. Una retícula en $\mathbf{V}$ generada por β, se define como el conjunto de todas las combinaciones lineales con elementos de β y escalares en $\mathbb{Z}$, denotada por Λ_β de la siguiente forma:

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

Se dice que β forma una $\mathbb{Z}$-base para la retícula. En general, diferentes bases de $\mathbf{V}$ generarán diferentes retículas. Aunque, si se considera la matriz de transición entre bases $[T]_{\beta\beta'}$ esta pertenece al grupo lineal general de $\mathbb{Z}$, es decir, $GL_m(\mathbb{Z})$. De esta forma, las retículas generadas por estas bases, serán isomorfas dado que $[T]_{\beta\beta'}$ induce un isomorfismo entre las dos retículas [12].

Definición 2.

Sea β una base de $\mathbf{V}$ un $\mathbb{R}$-espacio vectorial de dimensión finita y Λ_β una retícula en $\mathbf{V}$. Un dominio fundamental para Λ_β se define como [12]:

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

A. Problemas Fundamentales y Difíciles de las Retículas

Desde un punto de vista criptográfico, la conjeturada intratabilidad de tres problemas fundamentales y difíciles dentro de las retículas, son la base de la seguridad de los criptosistemas que operan en este tipo de estructuras. De ahí, que para aplicaciones, las retículas se toman en espacios vectoriales, a menudo $\mathbb{Q}^n$, o módulos libres, por lo regular $\mathbb{Z}^n$. Para ello, se debe considerar un espacio normado ($\mathbf{V}$, $||\quad||$) de dimensión n, y Λ como una retícula en $\mathbf{V}$, lo que permite definir lo siguiente: El problema del vector más corto (SVP, por sus siglas en inglés) tiene como finalidad encontrar el vecor no nulo más corto en Λ. El problema del vetor más cercano (CVP por sus siglas en inglés), para el cuál dado un vector $\mathbf{t} \in \mathbf{V}$, tal que no esté en Λ, su objetivo es encontrar un vector en Λ más cercano a $\mathbf{t}$. Por último, el problema de aproximación al vector más cercano ($apprCVP$ por sus siglas en inglés), para este, dado $\mathbf{t} \in \mathbf{V}$, se debe 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.

Para estos problemas, la *reducción de una retícula* es el nombre dado al problema práctico de resolver los problemas SVP y CVP. Es decir, encontrar vectores razonablemente cortos y bases *buenas* o más convenientes [13].

B. Bases Razonablemente Buenas

La idea de la reducción de la base recae en cambiar una base β de una retícula Λ en otra base más corta β' de tal manera que permanezca inalterada. Para hacer esto, se pueden usar las siguientes operaciones: Intercambio de dos vectores de la base. Reemplazo de v_j por $-v_j$ para sumar o restar a un vector v_j, una combinación lineal y discreta de los otros vectores de la base. Todas estas operaciones, sin que Λ se vea afectada [14].

C. Condiciones de Tamaño y Pseudo-Ortogonalidad

Suponiendo un conjunto linealmente independiente de vectores $\{v_1, ..., v_n\}$ y después del proceso de ortogonalización de Gram-Schmidth se obtiene un conjunto $\{v*_1, ..., v*_n\}$. Si algún coeficiente en dicho proceso satisface:

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

Al reemplazar v_i por $v_i - av_j$ con un a apropiado en $\mathbb{Z}$, se hace el coeficiente más pequeño. De ahí que se dice que una base satisface la *condición de tamaño* si:

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

ISSN 2395-8618

Para equilibrar esto, se requiere que los vectores base sean lo más ortogonales entre sí, por lo que debe imponer la *condición de pseudo-ortogonalidad*, donde:

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

De ahí que se tiene el Teorema de Hermite, deifnido en el Teorema 1. La demostración de dicho Teorema, puede consultarse en [15].

Teorema 1 (Hermite.): En cada retícula existe una base que satisface tanto la condición de tamaño como la condición de pseudo-ortogonalidad.

Desafortunadamente, los algoritmos más conocidos para encontrar *bases* son exponenciales en la dimensión. Por lo tanto, se debe cambiar la *condición de pseudo-ortogonalidad* a una menos escritcta, llamada la *condición de Lovász*:

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

La cuál indica que la proyección de v_{i+1} dentro del complemento ortogonal del subespacio generado por los vectores $v_1, ..., v_{i-1}$ es mayor o igual a tres cuartos de la proyección de v_i dentro del complemento ortogonal del subespacio generado por los vectores $v_1, ..., v_{i-1}$, lo que resulta ser generalización de la *condición de pseudo-ortogonalidad*, resultando en el Teorema 2. Su demostración puede verse en [15].

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.

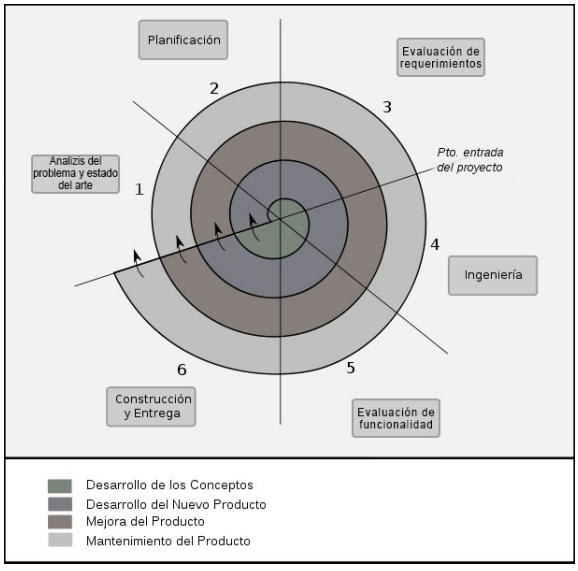


Fig. 1. Diagrama de metodología en utilizada para el diseño de la herrammienta para visualización

IV. Diseño de la herramienta

El desarrollo de la herramienta de *software* para la visualización de la criptografía basada en retículas considera el marco teórico de la Sección anterior dentro de la metodología de desarrollo de *software* en espiral, la cual permite tener una mejor fluidez, comunicación y división de trabajos [16]. De igual forma, permite empezar por funcionalidades básicas o generales, permitiendo mejorar y escalar el desarrollo de la herramienta. Dicha metodología se ilustra con la Figura 1.

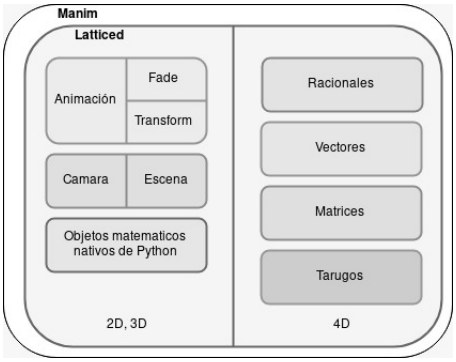


Fig. 2. Diagrama de bloques de la herramienta para visualización

A través de su uso, esta herramienta para visualización, además de implementar el algoritmo *LLL*, genera la gráfica de la retícula dada su base en un espacio euclideano, de tal forma que se tiene una herramienta para visualiza retículas en los espacios euclideanos de dimesiones dos, tres, cuatro y cinco (2D, 3D, 4D y 5D).

A. Bloques y módulos de la herramienta para visualización

Como se puede ver en la Figura 2, la herramienta se compone de dos bloques unidos por una interfaz gráfica (GUI), la cual permite escoger entre dos opciones, la primera opción para dimensión dos y tres (2D/3D), la segunda para dimensión cuatro y cinco (4D/5D).

Ambos bloques utilizan funciones y estructuras de la librería *Manim* [17] para graficar retículas degeneradas como no degeneradas, pero se diferencían por los módulos que utilizan para generar las animaciones. Para el caso de las retículas degeneradas, *Manim* da la posibilidad de graficar un dominio fundamental de cada una y en el caso de las retículas no degeneradas, permite aplicar el algoritmo de reducción LLL en dimensión dos y tres (2D/3D).

Las animaciones generadas para 2D/3D consideran tres módulos, el módulo de *Animación* que contiene a los submódulos *Fade* y *Transform*; el módulo de *Cámara*, que contiene al submódulo *Escena*; y por último el módulo de *Objetos nativos de Python*

El módulo de *Animación* se encarga de generar las transiciones *frame* a *frame* para la generación de la ecuación de la retícula. Los submódulos generan una animación suave, realizan algunos efectos sobre la animación y transforman los vectores visibles para la cámara. El módulo de *Cámara* se encarga de captar la proyección de la escena que se encuentra

ISSN 2395-8618

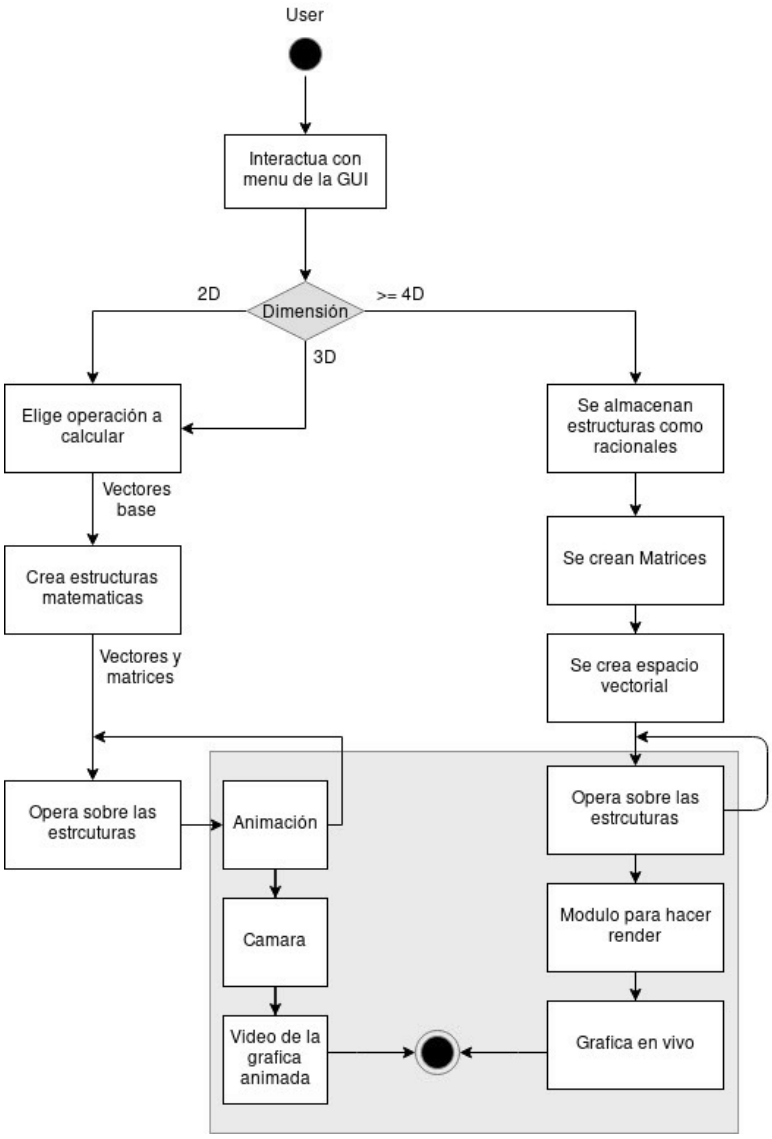


Fig. 3. Diagrama de procesos principales de la herramienta *latticed*

TABLA I
DATOS DE ENTRADA PARA LA PRUEBA DE FUNCIONALIDAD DE LAS DIMENSIONES 4 Y 5 (4D/5D)

Datos para las $\mathbb{Z}$-bases propuestas						
	Vectores	1D (u)	2D (u)	3D (u)	4D (u)	5D (u)
Retícula 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	-
Retícula no degenerada	v_1	0.5	0	0	0	0
	v_2	0	1	0	0	0
	v_3	0	0	1.5	0	0
	v_4	0	0	0	2	0
	v_5	0	0	0	0	2.5

en un espacio en 3D, realiza el corte del marco y genera una proyección en 2D para la generación de un video. Por último, el módulo de *Objetos nativos de Python* manipula los datos y realiza cálculos sobre los mismos.

Por otro lado, las animaciones generadas para 4D/5D se generan a través de la interacción de los módulos *Racionales, Vectores, Matrices* y *Tarugos*. Este bloque genera una animación o *render* que el usuario puede visualizar en

ISSN 2395-8618

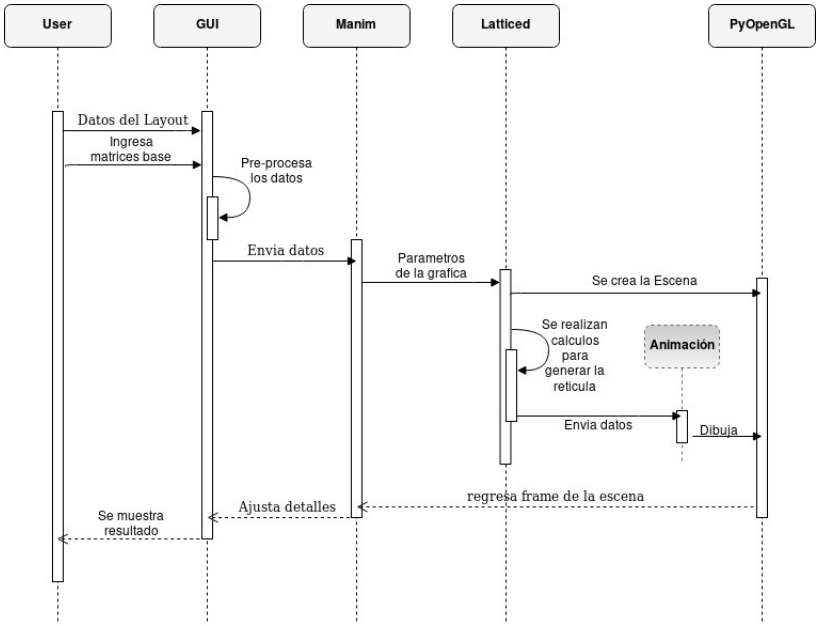


Fig. 4. Diagrama de secuencia de la herramienta *latticed*

TABLA II
COMPARACIÓN DE RESULTADOS OBTENIDOS

	Entradas				Salidas			
	Vectores	1D (u)	2D (u)	3D (u)	Vectores	1D (u)	2D (u)	3D (u)
Mathematica [7]	v_1	12	3	-	v_1	4	7	-
	v_2	4	7	-	v_2	8	-4	-
Herramienta de visualización	v_1	12	3	-	v_1	4	7	-
	v_2	4	7	-	v_2	8	-4	-
Mathematica [7]	v_1	13	2	5	v_1	-1	3	1
	v_2	12	5	6	v_2	-2	-2	1
	v_3	10	3	7	v_3	6	-1	9
Herramienta de visualización	v_1	13	2	5	v_1	-1	3	1
	v_2	12	5	6	v_2	-2	-2	1
	v_3	10	3	7	v_3	6	-1	9

tiempo real. Maneja datos de dimensiones superiores, por lo que se hace necesario contar con estructuras especializadas, de ahí que el módulo *Racionales* hace cálculos precisos, *Vectores* almacena la información en una estructura conocida, así como en los módulos de *Matrices* y *Tarugos*, estos últimos usando el metodo de triangulización.

De hecho, los *tarugos* de dimensiones cuatro y cinco aparecen cortados. Es decir, este módulo se diseño para que solo se dibuje una celda por *tarugo*, de tal forma que solo se genera el *render* de una cara del cubo. Esto, con el fin de ahorra poder de computo.

De ahí, que una retícula de dimensión cuatro se visualiza por celdas específicas de teseractos (cubos de dimensión cuatro) deformados por tranformaciones lineales determinadas por la $\mathbb{Z}$-base de la retícula. Lo mismo ocurre para la dimensión cinco, ya que la retícula se visualiza por cortes a los penteractos (cubos de dimensión cinco) y el sobrante de los cortes resultan ser teseractos deformados por transformaciones lineales determinadas por la $\mathbb{Z}$-base de la retícula. Lo anterior, haciendo operaciones y transformaciones a través de Python 3.7 y la biblioteca llamada PyOpenGL.

B. Flujo de actividades de la herramienta para visualización

El flujo de acciones de la herramienta considera la interacción entre los bloques y módulos con el usuario a través de una interfaz (GUI), lo cual se ilustra con la Figura 3. El flujo tiene inicio para seleccionar una opción, ya sea dimensión 2D/3D o 4D/5D. Para la opción 2D/3D se tiene el despliegue de un video con el cálculo y la respectiva animación. En caso de que el usuario seleccione la dimensión 4D/5D, la herramienta despliega como resultado un *render* que permitirá visualizar la gráfica generada.

Para ambas opciones, el usuario deberá introducir los datos de las retículas que se van a generar y calcular. Una vez que la herramienta cuente con estos datos, se organizan en estructuras especiales para poder operar sobre ellos. Cuando se realizan las operaciones, se entra a un bucle en el cual se generarán los *frames* para el *render*. Este bucle producirá los *frames* necesarios. Cabe destacar que durante este proceso se

ISSN 2395-8618

siguen realizando cálculos sobre los datos. Al final, se genera el respectivo resultado, el cual se muestra en pantalla para el usuario.

C. Secuencia de datos en la herramienta para visualización

La secuencia que sigue la herramienta para visualización tiene inicio cuando el usuario introduce datos para indicar cómo espera que se visualice el resultado. Estos, hacen referencia a la cámara y el tamaño, por mencionar algunos. Después, se ingresan los datos que permitirán generar las retículas, mismos sobre los que se hacen los cálculos. Estos datos se procesan para determinar la precisión y posteriormente ubicarlos en las estructuras adecuadas al tipo de dato que se introdujo de manera previa. Una vez que se tienen los datos estructurados, *Manim* y *Latticed* los operan de tal forma que se generan los *frames* que conforman el *render* final. Por último, PyOpenGL genera la visualización para el despliegue del resultado. Esto se ilustra con la Figura 4.

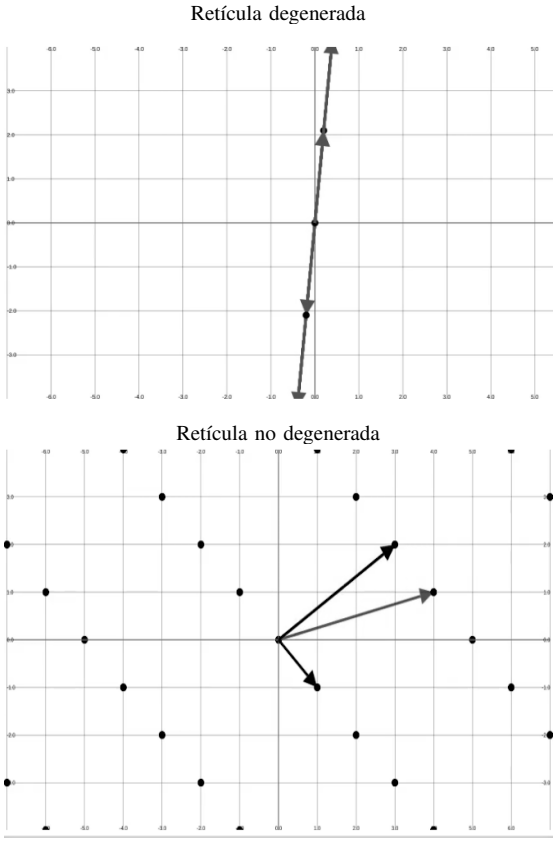


Fig. 5. Resultados obtenidos para las retículas en el espacio bidimensional

V. Pruebas y resultados

Las pruebas ejecutadas a la herramienta para visualización fueron de funcionalidad, las cuales se enfocaron en diferenciar principalmente la estructura utilizada, así como la dimensión del *render* [18]. La Tabla II y Tabla I presentan los datos de entrada para dichas pruebas. La Tabla II lista los vectores

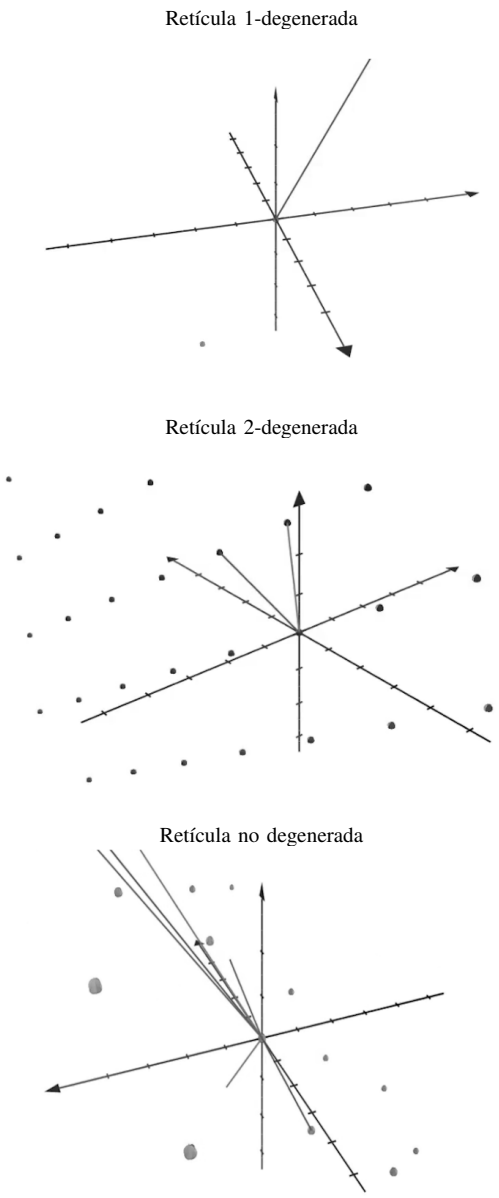


Fig. 6. Resultados obtenidos para las retículas en el espacio tridimensional

utilizados para las dimensiones 2 y 3 (D2/D3). La Tabla I lista los vectores utilizados para la dimensión 4 y 5 (D4/D5).

Los resultados obtenidos al ingresar los datos previamente listados, muestran la funcionalidad de la herramienta al momento de seleccionar, a través de la GUI, la opción referente a las dimensiones, ya sea 2D/3D o 4D/5D.

Después del escalonamiento del vector base y de la inserción de puntos, en la Figura 5 se puede ver la animación resultante para la dimensión 2 (2D). La Figura IV se muestra la animación resultante del vector $\mathbb{Z}-$base v_1 = (0.2, 2.1). En ese mismo sentido, la correcta implementación de los *tarugos* para la retícula no degenerada, así como del algoritmo de reducción

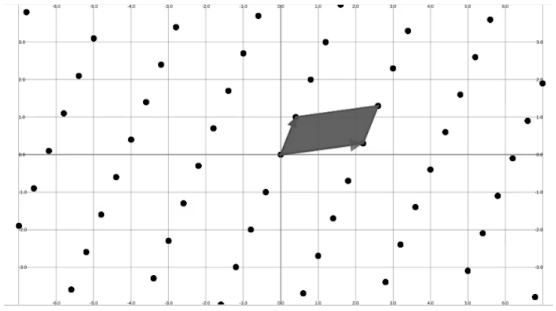
ISSN 2395-8618

LLL, permitió obtener la animación de la Figura IV, resultado de la $\mathbb{Z}$-base formada por los vectores v_1 = (12, 3) y v_2 = (4, 7).

La Figura 6 presenta los resultados de la animación para las retículas en el espacio tridimensional. La Figura IV muestra el vector $\mathbb{Z}-$base es v_1 = (2.3, 1.66, 5.4). En la Figura IV se puede ver que su $\mathbb{Z}$-base está formada por los vectores v_1 = (0.5, 1, 2.3), v_2 = (0, 3, 1). Finalmente, la animación de la Figura IV está dada por los vectores v_1 = (13, 2, 5), v_2 = (12, 5, 6), v_3 = (10, 3, 7). La obtención de las animaciones para la retícula 1 degenerada y la retícula 2 degenerada representan que la implementación de los *tarugos* es funcional, además de la correcta ejecución del algoritmo de reducción *LLL* a la retícula determinada por la $\mathbb{Z}$-base antes mencionada.

La Figura 7 muestra la animación obtenida para el dominio fundamental. La Figura V representa la animación resultante para el conjunto de vectores que forman parte de una $\mathbb{Z}-$base: v_1 = (0.4, 1), v_2 = (2.2, 0.3). Mientras que la Figura V muestra el resultado para la $\mathbb{Z}$-base que está formada por los vectores v_1 = (0.5, 2, 1), v_2 = (1, 3, 0.4), v_3 = (3, 1, 0.6).

Dominio Fundamental bidimensional



Dominio Fundamental tridimensional

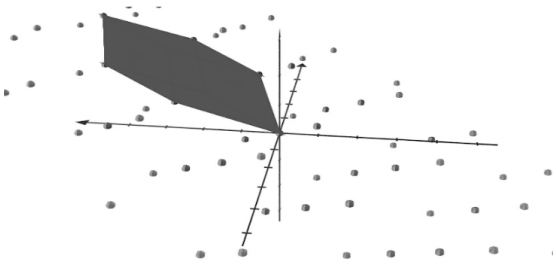


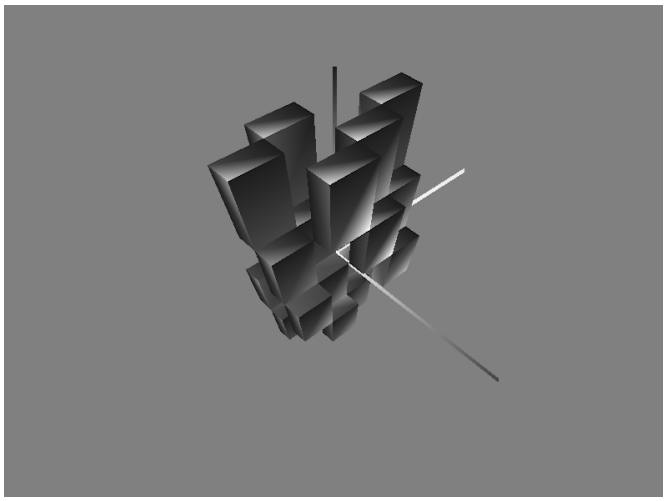
Fig. 7. Dominio Fundamental

Los resultados obtenidos para las dimensiones 4D/5D se muestran con la Figura 8, las cuales tomaron como entradas, la $\mathbb{Z}$-base, introducida a través de la GUI como enteros o racionales.

La Figura V muestra la animación resultante para la $\mathbb{Z}-$base 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).

La Figura V muestra la animación resultante para la $\mathbb{Z}$-base 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).

Retícula tetradimensional



Retícula pentadimensional

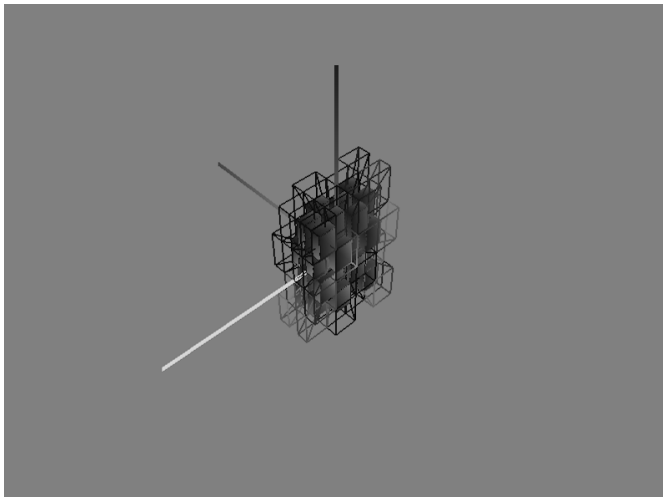


Fig. 8. Resultados obtenidos para las animaciones de retículas no degeneradas en dimensión 4D/5D

VI. Discusión

En los resultados obtenidos se muestra que la implementación del argoritmo *LLL* para reducir una retícula en las dimensiones dos, tres, cuatro y cinco (2D, 3D, 4D y 5D) la correcta funcionalidad, siempre y cuando los vectores introducidos no sean linealmente dependientes. En caso contrario, se efectua una división por cero dentro del algoritmo *LLL* y se notifica con la excepción: *try-except block*.

Un parámetro importante a considerar dentro de los resultados es la implementación de la teoría de los *tarugos* para optimizar los recursos computacionales a la hora de crear el *render*, ya que los resultados se observaron tanto en retículas degenerados como en no degeneradas. Esto dado que, entre más densa sea la retícula y mayor el número de elementos dentro del espacio de visualización, la herramienta necesita más tiempo para generar los *animaciones*.

ISSN 2395-8618

Esto, dado que la animación considera más elementos que mostrar en pantalla. Sin embargo, para las dimensiones 4D/5D se necesitó mucho mas tiempo para graficar correcamente el número de aristas y vértices que se incrementaron notoriamente en comparación con las dimensiones 2D/3D. Esto se debe a que entre más densa sea la retícula, y mayor número de elementos dentro del espacio de visualización, crece el tiempo de ejecución exponencialmente dentro a la herramienta, al terminar de animar, ya que debe animar más objetos en pantalla.

La correcta funcionalidad de la herramienta de visualización aquí presentada, se puede observar en la Tabla II en donde se presenta una comparación de los resultados obtenidos y *Mathematica*, una de las herramientas revisadas en el estado del arte.

VII. Conclusiones y trabajo futuro

La teoría detrás de la criptografía basada en retículas es de ende matemático e inspirado por el reino geométrico, de ahí que surge la necesidad de visualizar su fundamento, quedando de gran utilidad una herramienta para la visualización de la criptografía basada en retículas. De ella, se puede destacar la animación obtenida con diferentes dimensiones, 2D, 3D, 4D y 5D, dado que permite mostrar el diseño de la primera herramienta de *software* para graficación y animación de retículas de distintas dimensiones a través del uso de algoritmos de reducción y de la teoría de *tarugos*.

Esta herramienta ha explorado distintas dimensiones, de las cuales, las dimensiones cuatro y cinco (4D/5D), dado el crecimiento exponencialmente en el número de elementos, toma tiempo en generar la animación respectiva, siendo así, el motor de animación el que podría verse como una desventaja.

Como trabajo a futuro queda abierta la sugerencia de implementar el algoritmo *LLL* en dimensión cuatro y cinco de una manera visual y eficiente. Esto, cambiando a un lenguaje compilado en lugar de uno interpretado, como el que se utiliza en este trabajo, de forma tal que sea posible optimizar el tiempo de *renderización*.

Referencias

[1] P. W. Shor, .Algorithms for quantum computation: discrete logarithms and factoring,ïn *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994.

[2] J. Hoffstein, J. Pipher, and J. H. Silverman, *NTRU: A new high-speed public key cryptosystem*, Manuscript circulated at CRYPTO 1996 rump session, 1996.

[3] J. P. Buhler and P. Stevenhagen, "Algorithmic number theory. Lattices, number fields, curves and cryptography,Çambridge University Press, 2008.

[4] Y. Chen and P. Q. Nguyen, "BKZ 2.0: Better Lattice Security Estimates,ïn *International Conference on the Theory and Application of Cryptology and Information Security*, 2011.

[5] Microsoft, "Microsoft cryptographic technologies,"2018. https://docs.microsoft.com/en-us/windows/win32/seccrypto/cryptography-tools.

[6] Waterloo Maple Inc., *Maplesoft*, 2019. https://www.maplesoft.com/applications/view.aspx?SID=4498.

[7] Wolfram Research Inc., *Mathematica*, 2020. https://www.wolfram.com/?source=nav.

[8] H. Cohen, *PARI/GP*, 2018. https://pari.math.u-bordeaux.fr/.

[9] University of Sydney, *Magma Computational Algebra System*, 2010. http://magma.maths.usyd.edu.au/magma/.

[10] B. Esslinger, A. Wacker, and N. Kopal, *CrypTool 2*, 2018. https://www.cryptool.org/de/cryptool2.

[11] C. L. Siegel, *Lectures on the geometry of numbers*, Springer-Verlag, Berlin, 1989.

[12] J. Hoffstein, J. Pipher, and J. Silverman, *An Introduction to Mathematical Cryptography*, Springer, 2018.

[13] D. Simon, "Selected applications of LLL in number theory,"*Information Security and Cryptography*, 2009.

[14] A. K. Lenstra, H. W. Lenstra, and L. Lovász, "Factoring polynomials with rational coefficients,"*Mathematische Annalen* 513–534, 1982.

[15] D. J. Bernstein, J. Buchmann, and E. Dahmén, *Post-quantum cryptography*, Berlin: Springer, 2009.

[16] B. Boehm and W. J. Hansen, *Spiral Development: Experience, Principles and Refinements*, Carnegie Mellon University, 2000.

[17] G. Sanderson, *3blue1brown*, 2019, https://github.com/3b2b/manim.

[18] R. B. Gardner, *The Elements of Real Analysis*, Wiley and Sons, 1976.