Team Number:	IMMCHE00048
Problem Chosen:	B

Geometric Research on Sanxingdui Cultural Relics

Summary

The main point of this paper is to accurately describe, and estimate the curve equations of the bronze sun wheel and the complete surface and mass of the incomplete golden mask unearthed from the Sanxingdui archaeological site.

For question 1, after measurements on the given and collected images of the bronze sun wheel, pixel sizes at each part are obtained twice through in the form of proportion, which confirms that the **measurement error is no more than 3%** despite of different sources. Based on that, we respectively construct equations of concentric circles and arc lines corresponding to its five internal "rays", on the basis of **standard circle and hyperbola**. Via Python, we employ the Monte Carlo method to estimate the pixel area of the wheel to the circumscribed square in the form of proportion, and define the 'Coefficient of Similarity' to assess the **similarity**, which is verified to be as high as **97.56%**. Furthermore, we use these generalized equations to depict sun wheel patterns with 4, 6, 8, and 12 arcs.

For question 2, mirror recovery is initially utilized through OpenCV library and digital image processing techniques is then applied to repair the damaged parts to complete the whole gold mask. Taking the tip of the mask's nose as the origin, fourth-degree polynomials are used to fit equations for the 'double eye edge' and 'double ear edge' with points manually selected, which is validated through Canny edge detection algorithm, with a **99.54% overlap** between the automatically detected and manually fitted curves. The curve equations were presented. Finally, with the help of a 2D-3D scale conversion method, the area of the complete gold mask is scaled from its project, and estimated to be **1045.65** cm^2 , with its weight further assessed **563.34** g approximately, which aligns with expert estimation in relevant literature.

Our research offers a new perspective for the restoration of cultural relics, demonstrates the potential fusion of mathematics and archaeology and paves the way for the preservation and popularization of cultural heritage. The methods proposed herein can serve as useful references for the research and conservation of other ancient cultural sites.

Key words: polynomials regression, Canny edge detection algorithm, 2D-3D scale conversion

Catalogues

1 Problem Restatement

1.1 Backgrounds

Sanxingdui Archaeological Site, with a history of 3000 to 5000 years, is the largest, oldest, and most culturally rich ancient Shu culture site in the southwestern region of China. Regarded as one of the greatest archaeological discoveries of the 20th century, Sanxingdui not only reveals that both the Yangtze River Basin and the Yellow River Basin were cradles of Chinese civilization, but also unearths numerous artifacts of great historical, scientific, cultural, artistic, and aesthetic value.

These include the unique treasures of a 2.62-meter-high bronze Daliren, a 1.38-meter-wide bronze mask, and a 3.95-meter-high bronze divine tree. In addition, there are also unique precious items such as a golden staff and jade objects. Since the 1920s, Chinese and foreign archaeologists have conducted extensive excavations and specialized studies on the site, discovering city wall remains and a large number of exquisite cultural relics, including a half-faced gold mask. The mask is approximately 23 centimeters wide and 28 centimeters high, which strongly proves the existence of the ancient Shu kingdom four thousand years ago and the diversity of the origins of Chinese civilization. Here, against the backdrop of Sanxingdui, what we need to do is establish a mathematical model, collect relevant data, and solve problems stated below.

1.2 Problems

1.2.1 Bronze sun wheel

Among the relics, there is a bronze wheel discovered at Sanxingdui, with a radius of 85 centimeters, which resembles a modern steering wheel. To study the wheel, mathematical model should be first established. And the task is divided into three two steps, with the center of the bronze sun wheel set as the coordinate origin.

1. form the curve equations of the bronze sun wheel and the five internal "rays" corresponding to the arc lines.
2. draw bronze sun wheel patterns with four, six, eight, and twelve rays based on the general equation.

1.2.2 Gold Mask

From the relics, a half-faced gold mask was found. Its width is about 23 centimeters and its height is about 28 centimeters, which is currently the heaviest gold artifact discovered in China during the same period. The goal is threefold:

1. restore this half mask to a complete one.
2. calculate the curve equations corresponding to the "double eye edge" and "double ear edge" in the golden mask, with the nose tip of the golden mask set as the coordinate origin.
3. estimate the surface area and mass of the complete golden mask.

2 Hypotheses

1. The arc lines of the inner sun wheel can be considered tangent to the inner circle.
2. Assuming the arc line of the inner sun wheel corresponding to the 'rays' resemble hyperbola.
3. The camera is positioned directly in front of the wheel for the photograph.
3. Assuming a linear relationship between the area of the 2D projection of the gold mask and its 3D entity.
4. Assuming a linear relationship between the mass of the 2D projection of the gold mask and its 3D entity.

3 Description of Symbols

Symbol	Description
R_1	Radius of the inner small circle of the bronze sun
R_2	Radius of the outer circle of concentric ring in the bronze sun wheel
n	Number of 'rays' in the bronze sun wheel
ω	Width of the ring
f	Rotation matrix
θ	Angles between adjacent 'rays'
η	Coefficient of the similarity
$P_{overlap}$	Percentage of overlap
V	Volume of the mask
S	Area of the mask
M	Mass of the mask

4 Problem 1: Bronze Sun Wheel

4.1 Analyses

Through the raw image (left in Fig. 1) and what we collected from the official website (right [1] in Fig. 1), taking the center of the wheel as the coordinate origin, the bronze sun wheel is mainly composed of a concentric ring and an inner small circle, segmented by several "rays" evenly, with the curve of the 'rays' approximately viewed as a hyperbola, which is tangent to the inner circle and intersects with the ring. Therefore, the modeling of the sun wheel shape can be divided into three parts.

First, construct the equations for the concentric ring and the inner small circle, where the value of radius (or diameter) is the unique parameter of circles. For the 3rd hypothesis, data could be obtained through calculating pixel distances between sampled points. And to minimize manual sampling errors, we take the average of diameters resampled five times at different positions.

Second, build up equations for arc lines corresponding to the 'rays'. In general, polynomial curve fitting can be used to approximate curves by providing the coordinates of points. However, to reduce errors caused by manual data collection, and considering that the curve is tangent to the inner small circle as well as the distance from the tangent point to the center of the circle is constant, we suppose the curve of 'rays' as a part of hyperbola. And for a standard hyperbola symmetric to the x-axis, of which lengths of the real semi-axis and the imaginary semi-axis pertains to radius (R_1) and numbers of 'rays' (n), the curve of 'rays' could be formed while the standard hyperbola rotates around the coordinate origin by θ degrees.

Finally, to validate the reliability of our model, Monte Carlo algorithm is applied to estimate the area of both the bronze sun wheel in picture, and the area of wheel generated through the model.

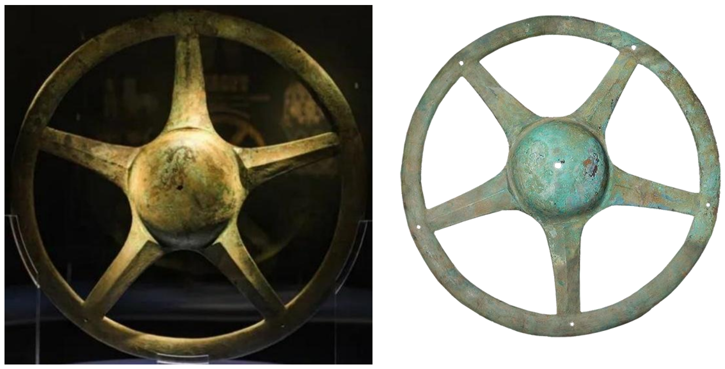
Fig. 1 Raw Image (left) and official Image of the unique wheel (right)

4.2 Modeling

4.2.1 Calculation of pixel distances

Improving the reliability of measurements to provide accurate data is crucial. Various errors may occur during this process, such as systematic errors and stochastic errors, related to methods, instruments and operations. To mitigate the impact of the stochastic errors, it is common and effective to replicate multiple times to get the average value, so as to reveal the overall characteristics of the sample.

Based on analyses in section 4.1, the model we constructed requires four parameters, including two diameters of the concentric ring, one diameter of the inner small circle, and the rotation angle, which pertains to the number of ‘rays’. So, what we only need to provide is diameters, which is actually turned into the diameter of the outer circle and the width of the ring, a more accurate calculation, compared to measuring two diameters respectively. Besides, for validating the model's efficacy, we also provide the minimum and maximum widths of the ‘rays’.

4.2.1.1 Pixel distances from two resources

For better precision, we sample on the two images of the sun wheel. To achieve this, we utilize the Python OpenCV library to obtain coordinates of points (in Fig. 2) and calculate the distance through Euclidean distance, with Table. 1 and Table. 2 for the raw image and the official one respectively (Appendix 8.1,8.2).

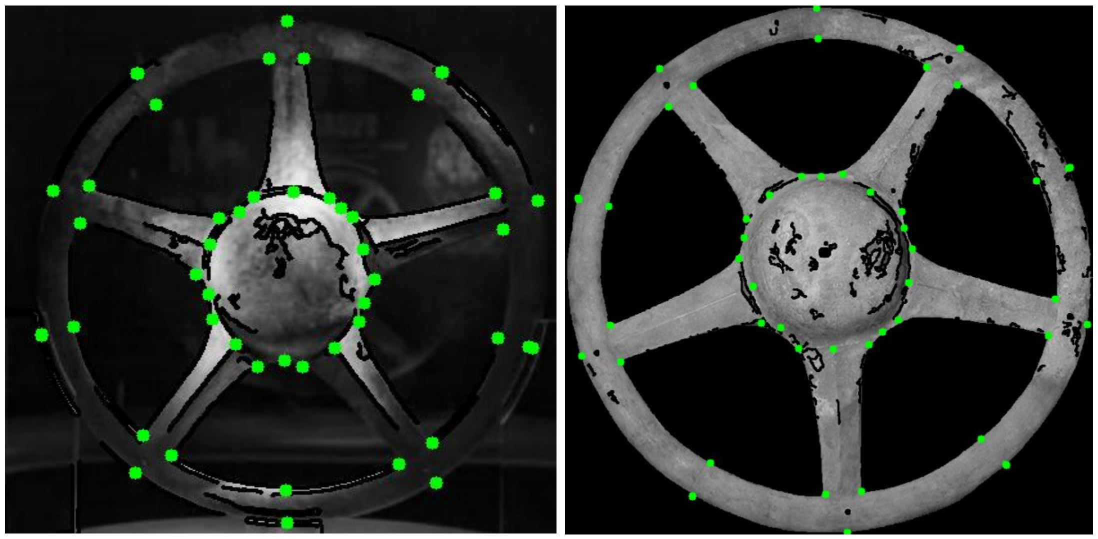

Fig. 2 Samples (left for raw image, right for the official one)

Table. 1 Measurements for the raw image

Times	w	R_2	R_1	Max_rays	Min_rays
1	25.71	402	131.75	48.47	30.81
2	26.02	393.71	136.4	51.66	27.2
3	25	400.2	128.88	56.85	31.06
4	25.46	401.76	135.18	52.81	29.61
5	29.15	400.82	125.28	59.01	27
Ave	26.26	399.69	131.49	53.76	29.14

Table. 2 Measurements for the official image

Times	w	R_2	R_1	Max_rays	Min_rays
1	41.05	700.21	231.55	93.41	49.68
2	42.45	702.20	228.04	91.08	50.51
3	46.24	699.94	216.54	93.49	49.82
4	50.12	704.41	217.17	96.88	48.17
5	47.38	697.83	237.26	94.13	50.96
Ave	45.448	700.918	226.112	93.798	49.828

4.2.1.2 Ratio for the consistency

It is obvious that the two images are of different size, which generates divergency for next steps. To deal with this problem, we introduce the **Ratio**, and use the largest parameters among the five as the reference, Diameter of the Outer circle, to accounting for the relative size of each other parameter. Results are shown in Table. 3, with the percentage of differences calculated by Eq. (1)

$$E = \frac{A-B}{A} \times 100\% \tag{1}$$

Where A represents the Ratio in the Raw image, and B represents ration in the official image.

Table. 3 Ratios and differences

	Raw image	Official Image	Differences (%)
$\frac{\text{Width of the ring}}{\text{Diameter of Outer\_circle}}$	0.0657	0.0648	-1.37
$\frac{Diamter\ of\ the\ inner\ small\ circl}{Diameter\ of\ the\ Outer\_circle}$	0.3290	0.3225	-1.97
$\frac{\text{Max width of the ring}}{\text{Diameter of the Outer\_circle}}$	0.1345	0.1338	-0.52
$\frac{\text{Min width of the ring}}{\text{Diameter of the Outer\_circle}}$	0.0729	0.0710	-2.60

From the differences above, it is seen that the difference between the two sets of ratios, does not exceed 3%, which suggests a strong consistency between the two pictures. Considering the various errors that may occur in measurements, such small differences are totally acceptable. Therefore, it could be concluded that the measurement from different sources, are both reliable. And since the official image is clearer than the other, it is thus further utilized in the following tasks.

4.2.2 Equations

In the Cartesian coordinate system, using the center of the bronze sun wheel as the coordinate origin, a circle would be described via Eq. (2).

$$x^2 + y^2 = R^2 \tag{2}$$

Where R is exemplified by R_1, and R_2.

For each ray, the equation could be easily described as Eq. (3).

$$y = kx \tag{3}$$

Where k is determined by θ, and as for two adjacent rays symmetric to x-axis, $\tan(\frac{\theta_i}{2}) = k$, ($\theta_i = i \times \theta, i = 1, 2, ..., n-1, \theta = \frac{2\pi}{n}$).

As what we assumed and analyzed before, the arc lines corresponding to 'rays' could be viewed as rotated hyperbolas. Since a standard hyperbola, which is symmetric to x-axis, could be described as Eq. (4), after rotation, the new coordinates $(x^{'}, y^{'})$ is thus derived based on the old coordinates (x, y) and a rotation matrix f in Eq. (5), which is uniquely defined by rotation degrees θ. The relationship through coordinates is shown via Eq. (6) (Code in Appendix 8.3). And through these equations, their patterns are shown in Fig. 3.

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1, a = R_1, b = R_1 \times \tan(\frac{\theta_i}{2}) \tag{4}$$

$$f = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{5}$$

$$\begin{pmatrix} x^{'} \\ y^{'} \end{pmatrix} = f \begin{pmatrix} x \\ y \end{pmatrix} \tag{6}$$

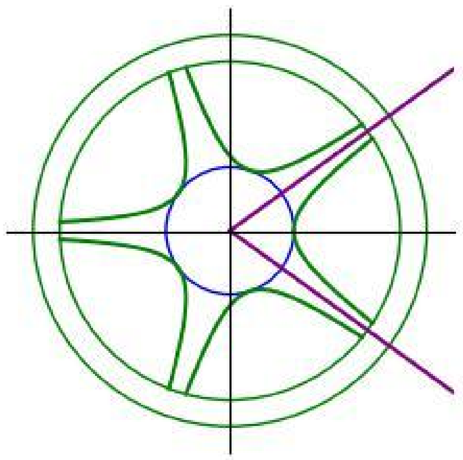
Fig. 3 Patterns of our model

4.3 Validation and Evaluation

According to steps above, we have established a model drawing patterns similar to the bronze sun wheel, but what is left is to quantify their similarity, and validate the efficacy of our model. So, the wheel with 5 ’rays’ is set as an example, we focus on the area of the wheel for comparisons between the given image and the pattern we draw.

4.3.1 Area and proportion of the wheel

To calculate the area of the wheel, Monte Carlo method[2-3] is selected for its simplicity in a complex system. Specifically, to estimate the area of a complex pattern, we can perform random sampling within a known area of a regular shape (such as a rectangle), and then calculate the proportion of points that fall within the target shape. So, for a bronze sun wheel, firstly, it is important to tell the wheel from its background. For the pattern produced by our model, it is dyed red (RGB=(255,0,0), and for the official image, we set the color of its background black (RGB=(0,0,0))[4]. After differing the pattern from its background, both of the picture is cropped to align the outer circle of the ring to the edge of the image. The above process is shown in Fig. 4 and Fig. 5.

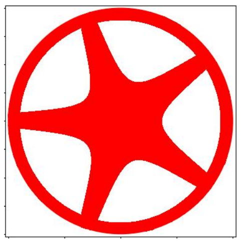
Fig. 4 red patterns of our model

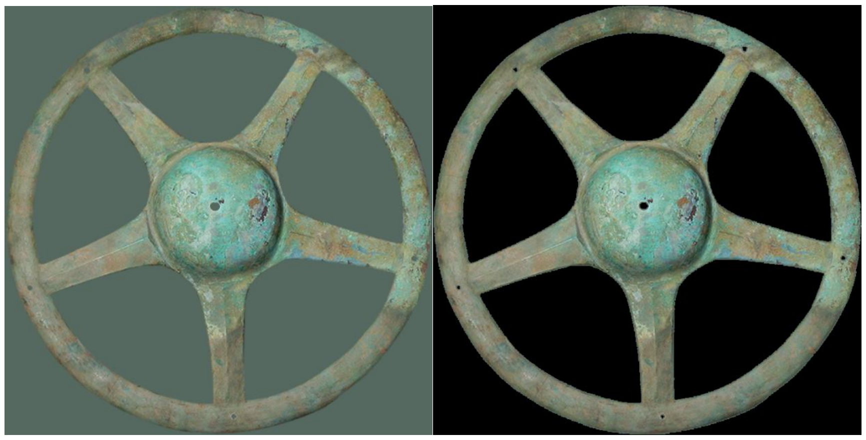
Fig. 5 clear wheel from the official image

Thus, the area of the wheels could be obtained through random sampling, and after a loop of 1e6 times, numbers of points located in the target pattern are obtained, with the pixel area of the picture shown in Table. 4, and code in Appendix 8.4-8.6.

Table. 4 Area and proportion of wheels from different sources

	Area of the whole picture	Area of the wheel	Approximate proportion
Official image	204634	499142	41%
Model pattern	52799.84	131769	40%

4.3.2 Coefficient of similarity

In order to more specifically evaluate the similarity between generated bronze sun wheel and the actual bronze sun wheel from the collected data, we design the 'Coefficient of similarity', to compare the proportions of the bronze sun wheel in Table. 4. To ensure η always falls within the range of 0 to 1, as well as to ensure that the more η is close to 1, the more similar they are, coefficient is defined as Eq. (7).

$$\eta = \frac{\min(P_{generated}, P_{actual})}{\max(P_{generated}, P_{actual})} \tag{7}$$

Where, $P_{generated}$ denotes the proportion of the model pattern and P_{actual} denotes the proportion of the official image.

4.4 Results

As we show before, $\eta = 40\% \div 41\% \times 100\% = 97.56\%$ is finally settled down through Eq. (7), which indicates a high resolution to the performance of our model. So, in conclusion, the mathematical model of the bronze sun wheel, is made up of the equation of the inner small circle and the concentric ring, $x^2 + y^2 = R^2$ where, R equals to

R_1^2, R_2^2 and $(R-w)^2$, and the curve equation of the arc lines corresponding to the 'rays', denoted by (x', y'), is described by $\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$, where (x, y) is restricted by $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1, a = R_1, b = R_1 \times \tan(\frac{\theta_i}{2})$, under the condition of $\sqrt{x'^2 + y'^2} < (R-w)$. And the patterns of the sun wheel with 4, 6, 8, and 12 'rays' are exhibited in Fig. 6, (see code in Appendix 8.7).

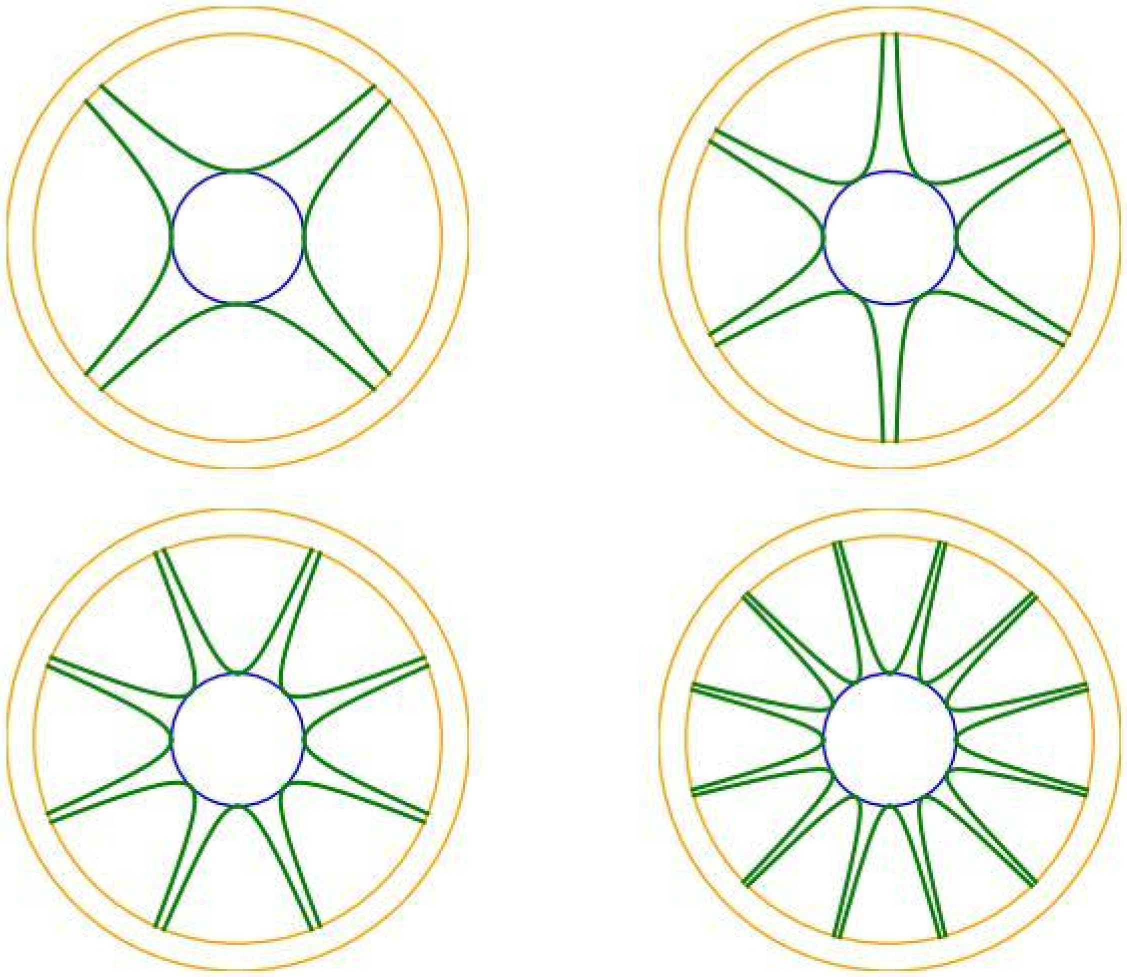

Fig. 6 Sun wheels with more internal "rays"

5 Problem 2: half Mask

5.1 Complete mask

In the field of archaeology, many artifacts discovered during excavations may be damaged or incomplete for various reasons. In order to visually reconstruct and present the complete form of these relics, researchers often consult literature and references to gather information. So, we first obtain a front-facing image of a mask through collecting data. And based on the image (Fig. 7) [5], through mirror recovery, digital image processing, manual drawing, and collected, the given mask is mirrored and repaired to be a complete one.

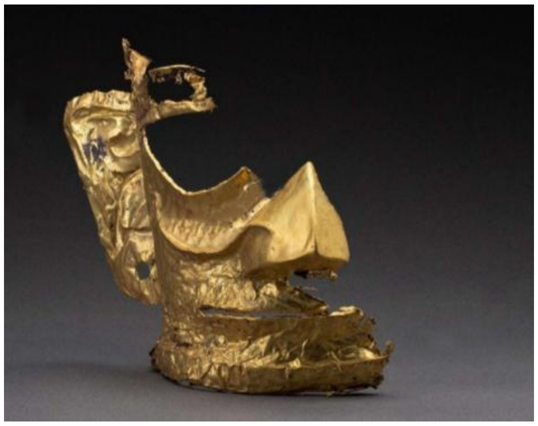

Fig. 7 A front face of the unique mask (collected)

5.1.1 Mirror recovery

To better repair the mask, a preprocess in necessary to ensure the quality of the following steps. Via Opencv library in Python [6], Fig. 7 is first transformed into RGB format. Since the mask is symmetric to the vertical axis, if the tip of its nose is set as the coordinate origin, we could flip it horizontally along the axis, to get a replicated two-faces mask. However, it is not enough, especially when two parts come into contact with each other, where could be visible seams or unnatural transitions. To address this issue, image blending techniques are employed. Specifically, we have chosen the weighted average method for blending [7], which considers predetermined weights to blend corresponding pixels from the two parts, ensuring smooth and continuous transitions from one side to the center axis. Additionally, the work by Burt and Adelson [8] provides useful methods and techniques in image blending, especially when dealing with high-frequency and low-frequency information. And the restored mask is shown in Fig. 8, with three parts, raw image at left, mirrored one at right, and blended area in the middle, included. (See Appendix 8.8)

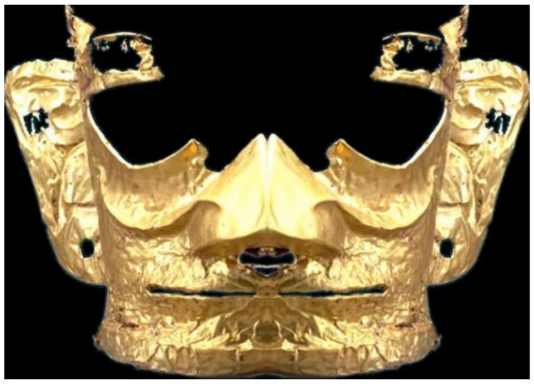

Fig. 8 roughly completed gold mask

5.1.2 Detailed repairment

Despite of the progress above, we still face the issue of missing details in the mask. To more accurately restore the original form of the gold mask, a comprehensive approach that combines digital image processing, manual drawing, and reference materials is employed, which allows for a more complete restoration and enhancement of the mask's appearance.

First, **digital image processing** through Photoshop [9] for basic repairing is an initial step. Patch-based techniques, particularly those that utilize information from surrounding pixels for restoration, have been proven effective in many studies. On the basis of that we have repaired minor damages in localized areas, such as the tip of the nose and upper parts of the ears. What's more, for areas contain complex structural information, such as subtle facial textures or concave contours in the mask, structured patching techniques and Poisson Blending [10] have been found to be highly suitable. So, considering color, texture, and structural information, coherence between the repaired sections and their surroundings is ensured. Further, while dealing with the loss of sculpted textures on the mask, texture synthesis techniques have proven to be the optimal choice. This technique effectively fills in textures by extracting them from other similar regions of the image

Second, **Hand painting** is particularly important for certain areas or artistic carvings on the gold mask that are difficult to restore using digital processing techniques, which would help preserve the original artistic style and structural integrity of the gold mask, and ensure continuity as well as consistency in the restoration process. By incorporating hand painting alongside digital processing, we can address the limitations of the latter and maintain the authenticity of the mask, which also allows for meticulous attention to details and the recreation of unique artistic carvings, ensuring that the restored areas seamlessly blend with the rest of the mask.

Finally, understanding the **historical and cultural background** of the gold mask is crucial when restoring it. By studying relevant historical documents and referencing other images with similar forms, shown in Fig. 9[1] , we can achieve a more accurate restoration of the missing structure and textures of the gold mask.

Overall, the combination of understanding the historical and cultural backgrounds, along with thorough research and reference analyses, contributes to a more precise restoration of the gold mask, capturing its true essence and maintaining its cultural heritage, which generates a more complete gold mask, close to the original form, compared to other relics, shown in Fig. 10.

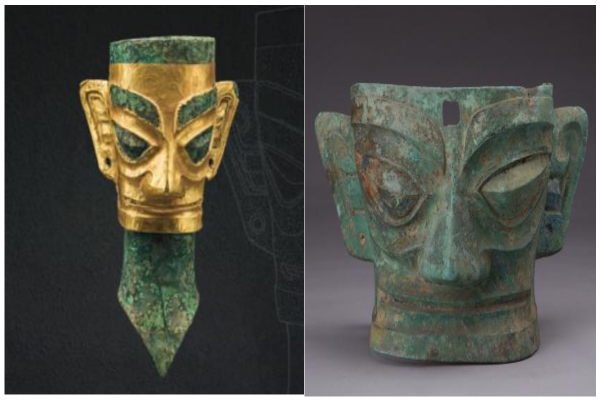
Fig. 9 Referencing other mask

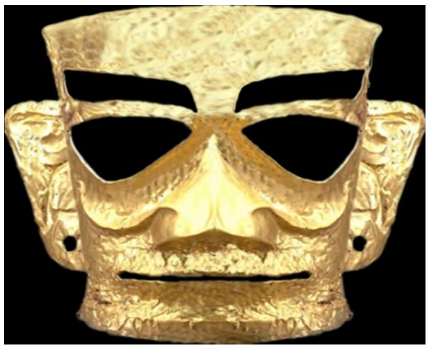
Fig. 10 Sophisticatedly restored mask

Compared to solely relying on hand painting or 3D software modeling, the integrated approach proposed in this paper offers several distinct advantages, including authenticity of restoration, operational efficiency, flexibility and consistency, as well as wide applicability. So combined the approach with the mirror recovery before hands, the comprehensive method indeed provides a complete and highly efficient restoration process.

5.2 Curve equations of the two edges

While dealing with complex archaeological artifacts like the Sanxingdui gold mask, it is crucial to provide accurate descriptions and models. In order to capture the geometric characteristics of the "double eye edge" and "double ear edge" of the gold mask, we have chosen a polynomial model for curve fitting. As described by Lawson and Hanson [11], polynomial fitting can provide a mathematical approximation for intricate geometric shapes, laying the foundation for subsequent analysis and computations.

5.2.1 Polynomial regression

To address the non-injective feature observed in certain areas of the gold mask, where some x-values correspond to two y-values, violating the injectivity property of a function, we decide to employ two polynomial curves to fit both of the ears and eyes. Taking the complexity of the mask's form and the precision requirements into account, the fourth-degree polynomial is chosen for regression, as Eq. (8).

$$y = ax^4 + bx^3 + cx^2 + dx + e \tag{8}$$

To figure out these parameters a, b, c, d, e , nonlinear least squares method is applied to minimize the differences between model values and observed values, with the objective function being the sum of squared residuals between observations and the model, which is shown in Eq. (9).

$$\min_{a,b,c,d,e} \sum_{i=1}^{n} [y_i - y(x_i)] \tag{9}$$

Where, n is the number of points.

5.2.2 Parameters of the equation

It is apparent that coordinates of points at the edge of ears and eyes are desperately required. So, taking the tip of the nose as the origin, a cartesian coordinate system is set for recording markers, for eight curves respectively. And in the paper, for brevity, only parts of samplings are shown through Fig. 11 and Fig. 12.

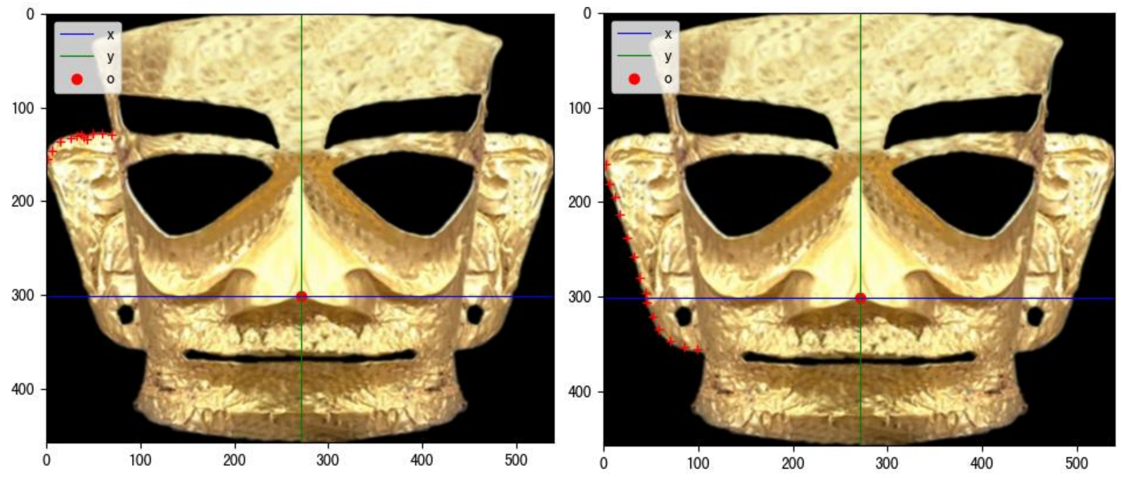

Fig. 11 Markers for the left ear

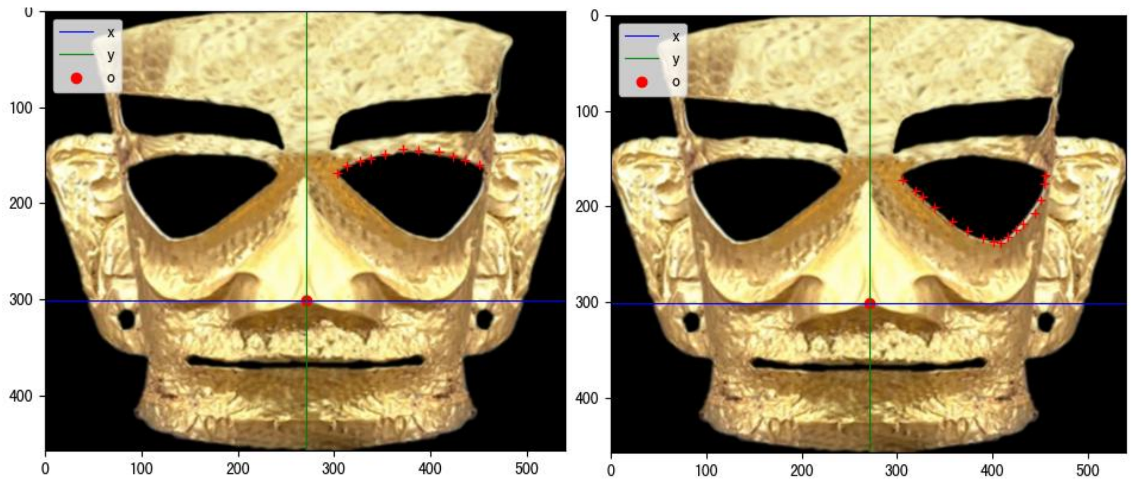

Fig. 12 Markers for the right eye

With records above, the optimize module from scipy library in Python is used for fitting the curve, via the curve_fit function. Equations of edges are listed as follows, with the precision decimally hind four. (code in Appendix 8.9)

For upper edge of the **left** eye, $y_1 = 4.7646e - 7x^2 + 0.0002x + 0.0330$

For lower edge of the left **eye**, $y_2 = -2.2992e - 7x^2 + -2.7289e - 5x - 0.0011$

For upper edge of the left **ear**, $y_3 = 1.2706e - 5x^2 + 0.0115x + 3.9439$

For lower edge of the left ear, $y_4 = 7.2356e - 6x^2 + 0.0060x + 1.8841$

For upper edge of the **right** eye,

$$y_5 = -6.8194e - 9x^2 + 7.6074x^3 + 0.0016x^2 - 0.6487x - 110.0319$$

For lower edge of the right **eye**,

$$y_6 = -7.4596e - 7x^4 + 0.0002x^3 - 0.0308x^2 + 2.6655x - 192.5417$$

For upper edge of the right **ear**,

$$y_7 = 1.0036e - 5x^4 - 0.0090x^3 + 3.0259x^2 - 450.3206x + 24882.8828$$

For lower edge of the right ear,

$$y_8 = 1.0105e - 5x^4 - 0.0084x^3 + 2.6226x^2 - 354.8813x + 17847.2580$$

5.2.3 Evaluation of regression

To test the accuracy of the manually selected curves, we compare them with edges detected by a well-acknowledged automatic method, Canny edge detection algorithm [12]. To visually show the differences, between the edges, we mark the completely overlapped curves with red markers, and biased parts with greed markers. And to further quantify the errors, a criterion named percentage of overlap is set as Eq. (10).

$$P_{overlap} = \frac{N_{green}}{N_{total}} \times 100\% \tag{10}$$

Where, N_{green} represents number of uncovered points, and N_{total} denotes the number of all selected points.

It seems feasible for most detection tasks, however, although some curves off track only one pixel remain acceptable in reality, they would be selected as a failure under the above criterion. To address this problem, we introduce a **searching window** [13] tolerating 3 pixels while detecting edges in surrounding areas, with results illustrated in Fig. 13.

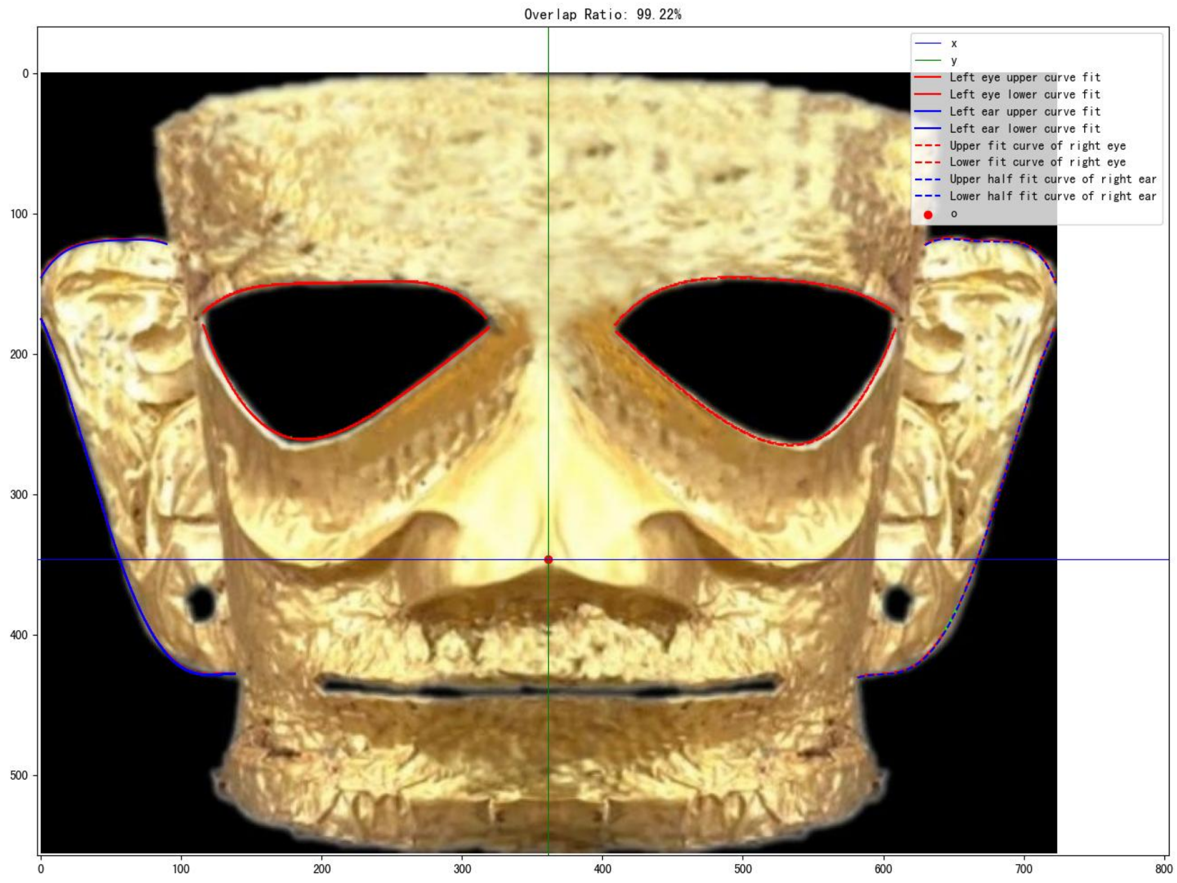

Fig. 13 Overlap between the manual curves and automatically detected curves

It could easily be found that the fitted curve is overlapped with edges detected by Canny edge detection method, where $P_{overlap}$ reaches 99.22%, an extremely prospective result, suggesting a precise capture of the edges and would strongly prove the accurate regression of corresponding equations.

Based on analyses above, our model provides a highly accurate description for specific regions of the gold mask. What's more, Canny edge detection and overlap calculation serve as compelling evidence for the efficacy of our model. Despite achieving good results, future research may consider incorporating more annotated data or adopting advanced curve fitting techniques to further improve the accuracy.

5.3 Estimation of the surface and the mass

5.3.1 2D-3D scale conversion

In the real world, there exists a relationship between the shape, area, and mass of a three-dimensional object and its projection in a two-dimensional image. Understanding this relationship can help us derive information about the three-dimensional object from its two-dimensional projection. However, this relationship is often complex and dependent on multiple variables such as the shape and viewing angle of the object. In this study, in order to simplify the problem, we assume linear relationships between the two-dimensional projection's area and the mass of the Sanxingdui gold mask, and its three-dimensional actual shape, as described in Section 2. Based on that hypothesis, Eq. (11) is presented.

$$M_{2D} = kM_{3D} \tag{11}$$

Where, M denotes the mass, and k is the key coefficient to transfer the mass of the 2D projection into 3D entity.

To get the value of k, we first need to quantify the 2D projection of the half mask, an image (Fig. 7) we collect, with the pixel size of 512×556 is set as the foundation, which holds the practical size of $23cm \times 28cm$. So, the ratios of width and the height ($Ratio_{width}, Ratio_{height}$) between 2D projection and the entity in 3D world could thus be calculated. And from the pixel area obtained by Monte Carlo method, similar to what we have done in Section 4.3.1, the pixel area of the 2D projection of the half-faced mask is thus 156053 (code in Appendix 8.10), which contributes to the derivation of real area of the 2D projection (S_{half_real})

$$Ratio_{width} = 23cm \div 512 pixel$$

$$Ratio_{height} = 28cm \div 556 pixel$$

$$S_{real\_half} = 156053 \times Ratio_{width} \times Ratio_{height} \approx 353.6cm^2$$

To further progress the mass of the mask in real world, the thickness is then required, after the real area, for the subsequent real volume. It is estimated through literature that the thickness of the gold mask varies from $0.2mm \sim 0.4mm$, and we thus settle down the value with the average of provided boundary, to be $0.3mm$. So, the real volume of the 2D projection is calculated $V_{real\_half} = 353.6cm^2 \times 0.03cm = 10.608cm^3$.

From the collected information [14], the composition of the half-faced gold mask is made up of the Gold, Silver and other materials, with density assumed as $19.32g/cm^3$, $10.49g/cm^3$, and $8g/cm^3$, accounting for 85%, 13.5% and 1.5% in volume respectively. And thus, the mass of the composition ($M_{\_real\_half}$) in 2D projection is listed and calculated below.

$$M_{2D\_real\_half} = 10.608cm^3 \times (85\% \times 19.32g/cm^3 + 13.5\% \times 10.49g/cm^3 + 1.5\% \times 8g/cm^3) = 191.14g$$

Since the real half-faced mask in reality is approximately 286g . k is thus concluded to be 0.6683, which means the mass of the 2D projection is 66.83% of the mass of the entity in 3D world.

By employing this approach, we have derived an initial two-dimensional to three-dimensional conversion coefficient, k .While our model is based on simplified assumptions, it still serves as a valuable tool for understanding and analyzing the actual shape and quality of the complete gold mask.

5.3.2 Scaling the complete mask

Similar to what we have done above, it is time to progress in estimating the complete mask. On the basis of restored image (Fig. 9), of which the pixel size is 704×596 . And the pixel area is also calculated through Monte Carlo method as 308902 (code in Appendix 8.11). So, the real area of the 2D projection of the complete mask ($S_{real\_complete}$) is thus equal to $698.81cm^2$, via $S_{real\_complete} = 308902 \times Ratio_{width} \times Ratio_{height} \approx 698.81cm^2$. And the area of the 3D entity would be obtained with the help of k, scaled as $1045.65cm^2$. Subsequently, the volume ($V_{3D\_real\_complete}$)and the mass ($M_{3D\_real\_complete}$) of the mask is going to be derived as follows.

$$V_{3D\_real\_complete} = 1045.65cm^2 \times 0.03cm = 31.37cm^3$$

$$M_{3D\_real\_complete} = V_{3D\_real\_complete} \times (85\% \times 19.32g/cm^3 + 13.5\% \times 10.49g/cm^3 + 1.5\% \times 8g/cm^3) = 563.34g$$

Apart from the results we conducted, according to the literature and expert speculation, the complete weight of this golden mask is presumed to be over $500g$ which aligns with our results, and further validate the accuracy of our mathematical models.

6 Conclusion and Prospects

6.1 Conclusion

This paper aims to model and evaluate artifacts from the Sanxingdui archaeological site, specifically the bronze sun wheel and the gold mask. Our methods combine traditional mathematical modeling, computer vision techniques, and data extrapolation to develop models that resonated with real-world conditions.

For the bronze sun wheel, the usage of concentric circles and arc lines provides a robust framework. The validation techniques, including Monte Carlo methods and image pixel ratio comparison, demonstrate a high fidelity with an accuracy of 97.56%. This serves as evidence of the efficacy of our approach and reinforces the feasibility of our mathematical model.

In the case of the incomplete gold mask, the mirror recovery and texture synthesis methods used for the artifact reconstruction are shown effective. Using the nose tip as the origin, the fourth-degree polynomial regression for the eye and ear edges showcases a high overlapping rate of 99.54% ,with the Canny edge detection results. This indicates the reliability of the curve fitting technique used in describing the mask's contour. Furthermore, the 2D-3D ratio conversion used for surface area and weight estimation aligned closely with expert estimates, reiterating the practicality and accuracy of the approach.

6.2 Prospects

Despite the progresses we have made, several avenues still emerge for further research.

1. Further Validation with Physical Measurements. While our models are prospective, their robustness can be further verified with physical measurements of the artifacts. Precision tools can measure intricate curves, validating our curve-fitting techniques.

2. Applying Advanced Machine Learning Techniques, Deep learning models, especially convolutional neural networks (CNN), could be trained to recognize and reconstruct damaged or partial artifacts, which might offer more accuracy and consistency over traditional image processing methods.

3. Historical Context Integration. The symbolic meaning of these artifacts still remains a mystery in many aspects. By correlating our models with historical records or

other archaeological discoveries, a more comprehensive understanding of the artifacts' significance might emerge.

4. Material Analysis Integration. By coupling our models with detailed metallurgical analysis, insights about the manufacturing techniques and aging processes of the artifacts can be deduced. This will further refine weight and structural estimates.

5. Digital Repository Creation: With the advent of augmented reality (AR) and virtual reality (VR), these high-fidelity models can be incorporated into digital museums, providing enhanced interactivity and understanding for enthusiasts and scholars alike.

In summary, while the current research has shed light on some of the intricate details and significance of the Sanxingdui artifacts, it also paves the way for more advanced and integrated approaches, combining the realms of archaeology, mathematics, and technology.

7 References

[1] SANXINGDUI MUSEUM . Available online : https://www.sxd.cn/relics/detail?id=16901972231329660 (accessed on 26/7/2023) , https://www.sxd.cn/relics/detail?id=16880327704677221

[2] Robert C P, Casella G, Casella G. Monte Carlo statistical methods[M]. New York: Springer, 1999.

[3] Caflisch, R. E. Monte Carlo and quasi-Monte Carlo methods. Acta Numerica, 7, 1-49. 1998.

[4] Szeliski R. Computer vision: algorithms and applications[M]. Springer Nature, 2022.

[5] SANXINGDUI MUSEUM . Available online : https://www.sxd.cn/relics/detail?id=16901972231329660 (accessed on 26/7/2023)

[6] Bradski G, Kaehler A. Learning OpenCV: Computer vision with the OpenCV library[M]. " O'Reilly Media, Inc.", 2008.

[7] Zitová, B., & Flusser, J. Image registration methods: a survey. *Image and vision computing*, 21(11), 977-1000. 2003.

[8] Burt, P.J., & Adelson, E.H. The Laplacian Pyramid as a Compact Image Code. *IEEE Transactions on Communications*, 31(4), 532-540.1983.

[9] Castleman K R. Digital image processing[M]. Prentice Hall Press, 1996.

[10] Pérez P, Gangnet M, Blake A. Poisson image editing[M]//ACM SIGGRAPH 2003 Papers. 2003: 313-318.

[11] Lawson C L, Hanson R J. Solving least squares problems[M]. Society for Industrial and Applied Mathematics, 1995.

[12] Canny, J., A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence, (6), pp.679-698. 1986.

[13] Wang, L., & He, D. C. Texture classification using texture spectrum. Pattern Recognition, 23(8), 905-910. 2008.

[14] Open the box and see the treasure! What are the six newly discovered pits in Sanxingdui Archaeology . Available online:

https://m.163.com/dy/article/G5INCAO20514D3UH.html (accessed on 26/7/2023).

8 Appendix

8.1 Image contour recognition in 4.2.1.1

```
import cv2

image = cv2.imread('1.jpg', cv2.IMREAD_GRAYSCALE)
blurred = cv2.GaussianBlur(image, (5, 5), 0)
edges = cv2.Canny(blurred, 50, 150)
contours,      _      =      cv2.findContours(edges,      cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(image, contours, -1, (0, 255, 0), 2)
cv2.imshow('Contours', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.imwrite('2.jpg', image)
```

8.2 Calculate pixel distance (need to choose reference point) in 4.2.1.1

```
import cv2
import numpy as np

points = []
def select_point(event, x, y, flags, param):
    global points
    if event == cv2.EVENT_LBUTTONDOWN:
        points.append((x, y))
        cv2.circle(image, (x, y), 5, (0, 255, 0), -1)
        cv2.imshow('Image', image)
        if len(points) == 2:
            distance   =   np.sqrt((points[1][0]   -   points[0][0])**2   +
(points[1][1] - points[0][1])**2)
            print(f"Pixel Distance: {distance:.2f}")
            points = []
image_path = "2.jpg"
image = cv2.imread(image_path)
cv2.namedWindow('Image')
cv2.setMouseCallback('Image', select_point)
cv2.imshow('Image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

8.3 Drawing a solar wheel through our model in 4.2.2

```
import matplotlib.pyplot as plt
import numpy as np
```

```
def plot_sun_wheel(n, r=180):
    plt.figure()
    circle = plt.Circle((0, 0), r, color='blue', fill=False)
    plt.gca().add_patch(circle)
    angles = np.linspace(0, 2*np.pi, n, endpoint=False)
    a = r
    b = r * np.tan(np.pi/n)
    t = np.linspace(-4*np.pi, 4*np.pi, 10000)
    outer_radius = 550
    inner_radius = 550-70
    for angle in angles:
        x = a * np.cosh(t)
        y = b * np.sinh(t)
        x_rot = x * np.cos(angle) - y * np.sin(angle)
        y_rot = x * np.sin(angle) + y * np.cos(angle)
        mask = (x_rot**2 + y_rot**2) < inner_radius**2
        x_rot = x_rot[mask]
        y_rot = y_rot[mask]
        plt.plot(x_rot, y_rot, 'g')
    outer_circle  =  plt.Circle((0,  0),  outer_radius,  color='orange',
fill=False)
    inner_circle  =  plt.Circle((0,  0),  inner_radius,  color='orange',
fill=False)
    plt.gca().add_patch(outer_circle)
    plt.gca().add_patch(inner_circle)
    plt.gca().set_aspect('equal', adjustable='box')
    plt.xlim([-560, 560])
    plt.ylim([-560, 560])
    plt.title(f"Sun Wheel with {n} lines")
    plt.show()
plot_sun_wheel(5)
```

8.4 Monte Carlo method for solving the generated solar wheel area in 4.3.1

```
import numpy as np
from PIL import Image

def is_red(pixel, threshold=100):
    red_dist = np.linalg.norm(np.array([255, 0, 0]) - np.array(pixel))
    return red_dist < threshold
image = Image.open("1.jpg")
image = np.array(image)
def monte_carlo_method(image, sample_size):
    height, width, _ = image.shape
    count = 0
    points = []
    for _ in range(sample_size):
        x = np.random.randint(0, width)
        y = np.random.randint(0, height)
        points.append((x, y))
        if is_red(image[y, x]):
            count += 1
    return points, count, count / sample_size
sample_size = 100000
points,   hit_red_count,   red_area_fraction   =   monte_carlo_method(image,
sample_size)
red_area = red_area_fraction * image.shape[0] * image.shape[1]
print("Number of red dots hit:", hit_red_count)
print("Estimated red pixel area:", red_area)
for point in points:
    image[point[1], point[0]] = [255, 255, 255]
marked_image = Image.fromarray(image)
marked_image.save("1_Marked.jpg")
```

8.5 Color the background of the solar wheel black in 4.3.1

```
from PIL import Image

def change_color(image_path, target_rgb, new_rgb, output_path):
    img = Image.open(image_path)
    img = img.convert('RGBA')
    data = img.getdata()
    new_data = []
    for item in data:
        if  item[0]  ==  target_rgb[0]  and  item[1]  ==  target_rgb[1]  and
item[2] == target_rgb[2]:
            new_data.append(new_rgb)
        else:
            new_data.append(item)
    img.putdata(new_data)
    img.save(output_path)
change_color("1.png", (86, 105, 96), (0, 0, 0, 255), "2.png")
```

8.6 Exhaustively calculating the pixel area of the solar wheel in 4.3.1

```
from PIL import Image

def count_pixels_not_equal_to_value(img_path, value=(0, 0, 0)):
    img = Image.open(img_path)
    img = img.convert('RGB')
    pixels = list(img.getdata())
    count = sum(1 for pixel in pixels if pixel != value)
    return count
def calculate_percentage(part, whole):
    return 100 * float(part) / float(whole)
```

```
img_path = "2.png"
total_pixels = Image.open(img_path).size[0] * Image.open(img_path).size[1]
non_target_pixels = count_pixels_not_equal_to_value(img_path)
print(f"Number of pixels not equal to (86, 105, 96): {non_target_pixels}")
print(f"Percentage            of            non-target            pixels:
{calculate_percentage(non_target_pixels, total_pixels):.2f}%")
```

8.7 Generate solar wheels with different number of arcs in 4.4

```
import matplotlib.pyplot as plt
import numpy as np

def plot_sun_wheel(n, r=180):
    plt.figure()
    circle = plt.Circle((0, 0), r, color='blue', fill=False)
    plt.gca().add_patch(circle)
    angles = np.linspace(0, 2*np.pi, n, endpoint=False)
    a = r
    b = r * np.tan(np.pi/n)
    t = np.linspace(-4*np.pi, 4*np.pi, 10000)
    outer_radius = 550
    inner_radius = 550-70
    for angle in angles:
        x = a * np.cosh(t)
        y = b * np.sinh(t)
        x_rot = x * np.cos(angle) - y * np.sin(angle)
        y_rot = x * np.sin(angle) + y * np.cos(angle)
        mask = (x_rot**2 + y_rot**2) < inner_radius**2
        x_rot = x_rot[mask]
        y_rot = y_rot[mask]
        plt.plot(x_rot, y_rot, 'g')
    outer_circle  =  plt.Circle((0,  0),  outer_radius,  color='orange',
fill=False)
    inner_circle  =  plt.Circle((0,  0),  inner_radius,  color='orange',
fill=False)
    plt.gca().add_patch(outer_circle)
    plt.gca().add_patch(inner_circle)
    plt.gca().set_aspect('equal', adjustable='box')
    plt.xlim([-560, 560])
    plt.ylim([-560, 560])
    plt.title(f"Sun Wheel with {n} lines")
    plt.show()
plot_sun_wheel(4)
plot_sun_wheel(6)
plot_sun_wheel(8)
plot_sun_wheel(12)
```

8.8 Golden Mask Mirror Restoration in 5.1

```
import cv2
import matplotlib.pyplot as plt

def blend_images(image1, image2, alpha=0.5):
    blended = cv2.addWeighted(image1, alpha, image2, 1 - alpha, 0)
    return blended
img_path = "2.png"
img = cv2.imread(img_path, cv2.IMREAD_UNCHANGED)
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
mirrored_img = cv2.flip(img_rgb, 1)
width = img_rgb.shape[1]
overlap_even_more_reduced = int(width * 0.01)
part_original_even_more = img_rgb[:, width-overlap_even_more_reduced:]
part_mirrored_even_more = mirrored_img[:, :overlap_even_more_reduced]
blended_part_even_more       =       blend_images(part_original_even_more,
part_mirrored_even_more)
final_mask_even_more_reduced            =            cv2.hconcat([img_rgb,
blended_part_even_more, mirrored_img[:, overlap_even_more_reduced:]])
plt.figure(figsize=(10, 5))
plt.imshow(final_mask_even_more_reduced)
plt.axis('off')
plt.show()
```

8.9 Interactive program for solving curve equations in 5.2

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
import cv2

img = plt.imread('1.jpg')
img_copy = np.copy(img)
plt.imshow(img)
print("Please click on the tip of the mask's nose as the origin")
nose_point = plt.ginput(1)
origin_x, origin_y = nose_point[0]
plt.scatter(origin_x, origin_y, c='red', marker='o', label="o")
plt.axhline(origin_y, color='blue', linewidth=0.8, label="x")
plt.axvline(origin_x, color='green', linewidth=0.8, label="y")
plt.legend(loc='upper left')
print("Please  click  on  the  upper  edge  of  the  left  eye,  press  enter  to
finish")
left_eye_up_points = plt.ginput(-1)
left_eye_up_x = [p[0] - origin_x for p in left_eye_up_points]
left_eye_up_y = [p[1] - origin_y for p in left_eye_up_points]
print("Please  click  on  the  lower  edge  of  the  left  eye,  press  enter  to
finish")
```

```
left_eye_down_points = plt.ginput(-1)
left_eye_down_x = [p[0] - origin_x for p in left_eye_down_points]
left_eye_down_y = [p[1] - origin_y for p in left_eye_down_points]
print("Please click on the upper edge of the left ear, press enter to
finish")
left_ear_up_points = plt.ginput(-1)
left_ear_up_x = [p[0] - origin_x for p in left_ear_up_points]
left_ear_up_y = [p[1] - origin_y for p in left_ear_up_points]
print("Please click on the lower edge of the left ear, press enter to
finish")
left_ear_down_points = plt.ginput(-1)
left_ear_down_x = [p[0] - origin_x for p in left_ear_down_points]
left_ear_down_y = [p[1] - origin_y for p in left_ear_down_points]
print("Please click on the upper edge of the right eye, press enter to
finish")
right_eye_up_points = plt.ginput(-1)
right_eye_up_x = [p[0] - origin_x for p in right_eye_up_points]
right_eye_up_y = [p[1] - origin_y for p in right_eye_up_points]
print("Please click on the lower edge of the right eye, press enter to
finish")
right_eye_down_points = plt.ginput(-1)
right_eye_down_x = [p[0] - origin_x for p in right_eye_down_points]
right_eye_down_y = [p[1] - origin_y for p in right_eye_down_points]
print("Please click on the upper edge of the right ear, press enter to
finish")
right_ear_up_points = plt.ginput(-1)
right_ear_up_x = [p[0] - origin_x for p in right_ear_up_points]
right_ear_up_y = [p[1] - origin_y for p in right_ear_up_points]
print("Please click on the lower edge of the right ear, press enter to
finish")
right_ear_down_points = plt.ginput(-1)
right_ear_down_x = [p[0] - origin_x for p in right_ear_down_points]
right_ear_down_y = [p[1] - origin_y for p in right_ear_down_points]
def func(x, a, b, c, d, e):
    return a * x**4 + b * x**3 + c * x**2 + d * x + e
left_eye_up_params, _ = curve_fit(func, left_eye_up_x, left_eye_up_y)
left_eye_down_params, _ = curve_fit(func, left_eye_down_x, left_eye_down_y)
left_ear_up_params, _ = curve_fit(func, left_ear_up_x, left_ear_up_y)
left_ear_down_params, _ = curve_fit(func, left_ear_down_x, left_ear_down_y)
x_vals = np.linspace(min(left_eye_up_x), max(left_eye_up_x), 1000)
y_vals = func(x_vals, *left_eye_up_params)
plt.plot(x_vals + origin_x, y_vals + origin_y, 'r-', label='Left eye upper
curve fit')
x_vals = np.linspace(min(left_eye_down_x), max(left_eye_down_x), 1000)
y_vals = func(x_vals, *left_eye_down_params)
```

```
plt.plot(x_vals + origin_x, y_vals + origin_y, 'r-', label='Left eye lower
curve fit')
x_vals = np.linspace(min(left_ear_up_x), max(left_ear_up_x), 1000)
y_vals = func(x_vals, *left_ear_up_params)
plt.plot(x_vals + origin_x, y_vals + origin_y, 'b-', label='Left ear upper
curve fit')
x_vals = np.linspace(min(left_ear_down_x), max(left_ear_down_x), 1000)
y_vals = func(x_vals, *left_ear_down_params)
plt.plot(x_vals + origin_x, y_vals + origin_y, 'b-', label='Left ear lower
curve fit')
plt.legend()
plt.show()
print(f"left_eye_up edge curve equation: y = {left_eye_up_params[0]}*x^2 +
{left_eye_up_params[1]}*x + {left_eye_up_params[2]}")
print(f"left_eye_down       edge       curve       equation:       y       =
{left_eye_down_params[0]}*x^2      +      {left_eye_down_params[1]}*x      +
{left_eye_down_params[2]}")
print(f"left_ear_up edge curve equation: y = {left_ear_up_params[0]}*x^2 +
{left_ear_up_params[1]}*x + {left_ear_up_params[2]}")
print(f"left_ear_down       edge       curve       equation:       y       =
{left_ear_down_params[0]}*x^2      +      {left_ear_down_params[1]}*x      +
{left_ear_down_params[2]}")
right_eye_up_params, _ = curve_fit(func, right_eye_up_x, right_eye_up_y)
right_eye_down_params,     _     =     curve_fit(func,     right_eye_down_x,
right_eye_down_y)
right_ear_up_params, _ = curve_fit(func, right_ear_up_x, right_ear_up_y)
right_ear_down_params,     _     =     curve_fit(func,     right_ear_down_x,
right_ear_down_y)
x_vals = np.linspace(min(right_eye_up_x), max(right_eye_up_x), 1000)
y_vals = func(x_vals, *right_eye_up_params)
plt.plot(x_vals  +  origin_x,  y_vals  +  origin_y,  'r--',  label='Upper  fit
curve of right eye')
x_vals = np.linspace(min(right_eye_down_x), max(right_eye_down_x), 1000)
y_vals = func(x_vals, *right_eye_down_params)
plt.plot(x_vals  +  origin_x,  y_vals  +  origin_y,  'r--',  label='Lower  fit
curve of right eye')
x_vals = np.linspace(min(right_ear_up_x), max(right_ear_up_x), 1000)
y_vals = func(x_vals, *right_ear_up_params)
plt.plot(x_vals  +  origin_x,  y_vals  +  origin_y,  'b--',  label='Upper  half
fit curve of right ear')
x_vals = np.linspace(min(right_ear_down_x), max(right_ear_down_x), 1000)
y_vals = func(x_vals, *right_ear_down_params)
plt.plot(x_vals  +  origin_x,  y_vals  +  origin_y,  'b--',  label='Lower  half
fit curve of right ear')
plt.legend()
plt.show()
print(f"right_eye_up edge curve equation: y = {right_eye_up_params[0]}*x^4
+    {right_eye_up_params[1]}*x^3    +    {right_eye_up_params[2]}*x^2    +
{right_eye_up_params[3]}*x + {right_eye_up_params[4]}")
print(f"right_eye_down      edge      curve      equation:      y      =
{right_eye_down_params[0]}*x^4    +    {right_eye_down_params[1]}*x^3    +
{right_eye_down_params[2]}*x^2     +     {right_eye_down_params[3]}*x     +
{right_eye_down_params[4]}")
print(f"right_ear_up edge curve equation: y = {right_ear_up_params[0]}*x^4
+    {right_ear_up_params[1]}*x^3    +    {right_ear_up_params[2]}*x^2    +
{right_ear_up_params[3]}*x + {right_ear_up_params[4]}")
print(f"right_ear_down      edge      curve      equation:      y      =
{right_ear_down_params[0]}*x^4    +    {right_ear_down_params[1]}*x^3    +
{right_ear_down_params[2]}*x^2     +     {right_ear_down_params[3]}*x     +
{right_ear_down_params[4]}")
def generate_model_points(params, x_range):
    x_vals = np.linspace(x_range[0], x_range[1], 2000)
    y_vals = func(x_vals, *params)
    points  =  [(int(x  +  origin_x),  int(y  +  origin_y))  for  x,  y  in
zip(x_vals, y_vals)]
    return points
gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
edges = cv2.Canny(gray_img, 30, 240)
all_model_points        =        (generate_model_points(left_eye_up_params,
(min(left_eye_up_x), max(left_eye_up_x))) +
                    generate_model_points(left_eye_down_params,
(min(left_eye_down_x), max(left_eye_down_x))) +
                    generate_model_points(left_ear_up_params,
(min(left_ear_up_x), max(left_ear_up_x))) +
                    generate_model_points(left_ear_down_params,
(min(left_ear_down_x), max(left_ear_down_x))) +
                    generate_model_points(right_eye_up_params,
(max(right_eye_up_x), min(right_eye_up_x))) +
                    generate_model_points(right_eye_down_params,
(max(right_eye_down_x), min(right_eye_down_x))) +
                    generate_model_points(right_ear_up_params,
(max(right_ear_up_x), min(right_ear_up_x))) +
                    generate_model_points(right_ear_down_params,
(max(right_ear_down_x), min(right_ear_down_x))))
def check_nearby(x, y, radius=3):
    for i in range(-radius, radius + 1):
        for j in range(-radius, radius + 1):
            if 0 <= y + i < edges.shape[0] and 0 <= x + j < edges.shape[1]:
                if edges[y + i, x + j] > 0:
                    return True
    return False
overlap_count = 0
for x, y in all_model_points:
    if 0 <= y < edges.shape[0] and 0 <= x < edges.shape[1]:
        if check_nearby(x, y):  # Use the new checking function
            img_copy[y, x] = [255, 0, 0]
            overlap_count += 1
        else:
            img_copy[y, x] = [0, 255, 0]
overlap_ratio = overlap_count / len(all_model_points)
print(f"Overlap Ratio: {overlap_ratio:.2%}")
plt.imshow(img_copy)
plt.title(f"Overlap Ratio: {overlap_ratio:.2%}")
plt.show()
```

8.10 Solve the pixel area of half a gold mask in the projection direction of the photo in 5.3.1.2

```
from PIL import Image

img = Image.open("1.jpg")
pixels = img.load()
black_count = 0
for x in range(img.width):
    for y in range(img.height):
        if pixels[x, y] == (0, 0, 0):
            black_count += 1
mask_count = img.width * img.height - black_count
print(f"The pixel area of the gold mask is: {mask_count} Pixel^2")
```

8.11 Solve the pixel area of a complete gold mask repaired in the projection direction of a photo in 5.3.2

```
from PIL import Image

img = Image.open("1.jpg")
pixels = img.load()
black_count = 0
for x in range(img.width):
    for y in range(img.height):
        if pixels[x, y] == (0, 0, 0):
            black_count += 1
mask_count = img.width * img.height - black_count
print(f"The 2D projected pixel area of the repaired complete golden mask 
is: {mask_count} Pixel^2")
```