Reconstruction
Code du TIPE

1 Code sur la première partie

1.1 Code du fichier

Ici, on insère le fichier entier, tel quel, pas forcément adapté.

```c
#include "triangle.h"

// Produit scalaire pour des vecteurs de dimension 3
double scalaire(double* v1, double* v2) {
    double s = 0;
    for (int i = 0; i < 3; i++) {
        s += v1[i] * v2[i];
    }
    return s;
}
double norme(double* v) {
    return sqrt(v[0] * v[0] + v[1] * v[1] + v[2] * v[2]);
}
// Addition de deux vecteurs dans v3
void add(double* v1, double* v2,double* v3) {
    for (int i = 0; i < 3; i++) {
        v3[i] = v1[i] + v2[i];
    }
}

// Addition in-place de deux vecteurs
void addin(double* v1, double* v2) {
    for (int i = 0; i < 3; i++) {
        v1[i] += v2[i];
    }
}

// Soustraction de deux vecteurs dans v3
void sub(double* v1, double* v2, double* v3) {
    for (int i = 0; i < 3; i++) {
        v3[i] = v1[i] - v2[i];
    }
}

// Mise à l'échelle d'un vecteur
void scale(double* v1, double s) {
    for (int i = 0; i < 3; i++) {
        v1[i] *= s;
    }
}

// Calcul du barycentre
void barycentre(triangle l, double* bary, double** points) {
    bary[0] = 0;
    bary[1] = 0;
    bary[2] = 0;
    addin(bary, points[l.a]);
```

```
    addin(bary, points[l.b]);
    addin(bary, points[l.c]);
    scale(bary, 1.0 / 3);
}

// Produit vectoriel pour des vecteurs de dimension 3
void prod(double* v1, double* v2, double* v3) {
    for (int i = 0; i < 3; i++) {
        v3[i] = v1[(i + 1) % 3] * v2[(i + 2) % 3] - v1[(i + 2) % 3] * v2[(i + 1) % 3];
    }
}

// Test si un vecteur est nul
bool nulv(double* v) {
    for (int i = 0; i < 3; i++) {
        if (v[i] != 0) {
            return false;
        }
    }
    return true;
}

// Calcul du nombre de combinaisons de 3 parmi n
unsigned long int trois_parmi(int n) {
    return (unsigned long int) (n <= 2) ? 0 : (unsigned long int)n * (unsigned long int)(n - 1) * (unsigned long int)(n - 2) / 6;
}

// Génère toutes les combinaisons possibles de triangles
triangle* triangles(int card) {
    unsigned long int n = trois_parmi(card);
    fprintf(stdout, "binomial(%d, 3) = %lu\n", card, n); fflush(stdout);
    triangle* trigs = malloc(n * sizeof(triangle));
    unsigned long int ind = 0;
    for (unsigned long int i = 0; i < card; i++) {
        for (unsigned long int j = i + 1; j < card; j++) {
            for (unsigned long int k = j + 1; k < card; k++) {
                trigs[ind].a = i;
                trigs[ind].b = j;
                trigs[ind].c = k;
                ind++;
            }
        }
    }
    return trigs;
}

// Impression d'un vecteur dans un fichier
void file_print_vect(double* v, FILE* file) {
    fprintf(file, "%f %f %f\n", v[0], v[1], v[2]);
}
```

```c

// Impression d'un vecteur sur la console
void print_vect(double* v) {
    printf("(%1.3f, %1.3f, %1.3f)\n", v[0], v[1], v[2]);
}

// Destruction des triangles
void destroy_trigs(triangle* l) {
    free(l);
}

// Destruction des points
void destroy_points(double** l, int n) {
    for (int i = 0; i < n; i++) {
        free(l[i]);
    }
    free(l);
}

bool* keeptrig(triangle* l, unsigned long int ntrig, int size, double** point)
 ↪  {
    printf("malloc bool debut\n");
    fflush(stdout);
    bool* res = malloc(ntrig * sizeof(bool));
    printf("malloc bool fin\n");
    fflush(stdout);

    double v1[3];
    double v2[3];
    double v[3];
    double n[3];
    double bary[3];

    for (unsigned long int i = 0; i < ntrig; i++) {
        sub(point[l[i].b], point[l[i].a],v1);
        sub(point[l[i].c], point[l[i].a],v2);

        prod(v1, v2,n);

        barycentre(l[i], bary, point);
        res[i] = true;

        int signe = 0;
        for (int j = 0; j < size; j++) {
            sub(bary, point[j], v);
            double s = scalaire(n, v);

            if (fabs(s) < 0.01) {
                s = 0;
            }
            if (s != 0) {
                if (signe == 0) {
```

```
                    signe = (s > 0) ? 1 : -1;
                } else if (signe * s < 0) {
                    res[i] = false;
                    break;
                }
            }
        }
    }
    printf("keeptrig fin calcul\n");
    fflush(stdout);

    return res;
}


// Lecture des points depuis un fichier
double** read_points(char* filename, int count) {
    char complete_fn[256];
    snprintf(complete_fn, sizeof(complete_fn), "points/donnees/%s.txt",
    filename);
    FILE* file = fopen(complete_fn, "r");
    if (!file) {
        perror("Erreur d'ouverture du fichier de points");
        exit(EXIT_FAILURE);
    }
    double** points = malloc(count * sizeof(double*));
    if (!points) {
        perror("Erreur d'allocation pour les points");
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < count; i++) {
        points[i] = malloc(3 * sizeof(double));
        if (!points[i]) {
            perror("Erreur d'allocation pour un point");
            exit(EXIT_FAILURE);
        }
        fscanf(file, "%lf %lf %lf", &points[i][0], &points[i][1],
    &points[i][2]);
    }
    fclose(file);
    return points;
}

double** rand_points(int n){
  double ** res = malloc(n*sizeof(double*));
  for(int i = 0; i<n; i++){
    res[i]=malloc(sizeof(double)*3);
    for(int j = 0; j< 3; j++){
      res[i][j] = rand()%1000*0.001;
    }
  }
  return res;
```

```c
}

void stl_generate(char* filename, double** point, triangle* l, unsigned long int ntrig,bool *garde){
    char complete_fn[256];
    snprintf(complete_fn, 256, "stltest/%s", filename);

    printf("ouverture stl fichier\n");
    fflush(stdout);
    FILE* file = fopen(complete_fn, "w");
    assert(file != NULL);

    printf("ecriture debut\n");
    fflush(stdout);
    fprintf(file, "solid \n");
    double v1[3];
    double v2[3];
    double n[3];
    int count=0;
    for (unsigned long int i = 0; i < ntrig; i++) {
      if (garde[i]){
        count++;
        sub(point[l[i].b], point[l[i].a],v1);
        sub(point[l[i].c], point[l[i].a],v2);

        prod(v1, v2,n);
        fprintf(file, "    facet normal %lf %lf %lf\n        outer loop\n",n[0],n[1],n[2]);
            fprintf(file, "        vertex %lf %lf %lf\n", point[l[i].a][0],point[l[i].a][1],point[l[i].a][2]);
            fprintf(file, "        vertex %lf %lf %lf\n", point[l[i].b][0],point[l[i].b][1],point[l[i].b][2]);
            fprintf(file, "        vertex %lf %lf %lf\n", point[l[i].c][0],point[l[i].c][1],point[l[i].c][2]);
        fprintf(file, "        endloop\n    endfacet\n");
      }
    }
    printf("nb triangles : %d\n", count);
    fprintf(file, "endsolid \n");
    fclose(file);
    printf("stl generated : %s\n", filename);
    fflush(stdout);
}

double** mat_to_table (matrice* mat,int* n){
  *n = mat->n;
  double** space = malloc(sizeof(double*)*(*n));
  for(int i = 0; i<*n;i++){
    space[i]=malloc(sizeof(double)*3);
    for(int j = 0; j<3;j++){
      space[i][j]=mat->mat[i][j];
    }
```

```
246     }
247     return space;
248 }
249 
```

1.2 Code du fichier `nom_fichier.c`

2 Code de la deuxième partie

2.1 Code du fichier

```cpp
#include "triangle.h"

// Produit scalaire pour des vecteurs de dimension 3
double scalaire(double* v1, double* v2) {
    double s = 0;
    for (int i = 0; i < 3; i++) {
        s += v1[i] * v2[i];
    }
    return s;
}
double norme(double* v) {
    return sqrt(v[0] * v[0] + v[1] * v[1] + v[2] * v[2]);
}
// Addition de deux vecteurs dans v3
void add(double* v1, double* v2,double* v3) {
    for (int i = 0; i < 3; i++) {
        v3[i] = v1[i] + v2[i];
    }
}

```